Temporal Model-Driven Systems Engineering

Cumulative Habilitation Thesis for Applied Computer Science at the Faculty of Social Sciences, Economics and Business of the Johannes Kepler University Linz
Abstract

Due to the paradigm shift towards Industry 4.0, the role of software-intensive systems is becoming more and more important. In particular, the trend towards physical components being controlled by software has led to the Internet-of-Things (IoT) and Cyber-Physical-Systems (CPS). As a consequence, companies face highly complex systems that are undergoing a constant change process resulting from shorter innovation cycles and rapidly changing customer needs. It is important that they keep their high-level requirements organized and consistent over multiple revision cycles across the entire life cycle of such a system, i.e., from design over development to implementation and operation.

Modeling is considered as a promising technique to better understand the dependencies within such complex systems. By following the Model-Driven Engineering (MDE) paradigm, systems are developed on a higher level of abstraction, and therefore, models are used as an integral part covering requirements, analysis, design, implementation, and verification. Although the term “model-integrated computing” has been coined almost twenty years ago, it has to be emphasized that the integration of models in the system life cycle is still mainly concerned with forward engineering, i.e., the development of new systems through generative techniques. Much less effort in MDE is spent on the evolutionary aspects of systems changing over time. For tackling this issue, models must no longer be considered as isolated one-shot system prescriptions, but as evolutionary and reusable descriptions of reality.

The research scope of this cumulative habilitation thesis is explicitly addressing this evolutionary aspect by focusing on temporal aspects of models of CPS. It follows a Model-Driven Systems Engineering (MDSE) approach by identifying and integrating appropriate concepts, languages, techniques, and tools for the systematic adoption of models throughout the engineering process. Models are continuously revised, often by considering feedback from other resources, until they are released. However, also the feedback after the release, i.e., from the operation, is reflected in the models. In the first part of this cumulative habilitation thesis, we elaborate on the integration of data from heterogeneous sources in order to provide a homogenized meaningful stack of information from the running system to a higher level of abstraction. In the second part, we cover the evolutionary aspects of engineering artefacts, i.e., models. Thereby, the focus is not only to represent the current state to steer the system, but on the representation of the system’s history. In the final part, we provide MDE techniques for analyzing runtime data and extracting descriptive models for reasoning about and validating the operation of systems.
## Contents

1 Introduction ................................................................. 1
   1.1 Motivation ............................................................... 1
   1.2 Background ............................................................ 2
   1.3 On the Need for Temporal Model-Driven Systems Engineering 5
   1.4 Challenges of Temporal Model-Driven Systems Engineering 6
   1.5 Contributions to Temporal Model-Driven Systems Engineering 10
   1.6 Summary ................................................................. 31
   1.7 Outlook ................................................................. 33
   1.8 Bibliography ............................................................ 34

2 From business functions to control functions: .......................... 51
   Transforming REA to ISA-95 ............................................. 51
   A. Mazak and C. Huemer
   DOI: 10.1109/CBI.2015.50

3 AutomationML, ISA-95 and Others: ...................................... 63
   Rendezvous in the OPC UA Universe .................................. 63
   B. Wally, C. Huemer, A. Mazak and M. Wimmer
   DOI: 10.1109/COASE.2018.8560600

4 Leveraging integration facades for model-based tool interoperability . 71
   G. Paskaleva, A. Mazak-Huemer, M. Wimmer and T. Bednar
   DOI: 10.1016/j.autcon.2021.103689
<table>
<thead>
<tr>
<th>Number</th>
<th>Title</th>
<th>Authors</th>
<th>Details</th>
</tr>
</thead>
</table>
12 Reverse Engineering of Production Processes based on Markov Chains

A. Mazak, P. Patsuk-Bösch and M. Wimmer
DOI: 10.1109/COASE.2017.8256182

13 Automatic Reverse Engineering of Interaction Models from System Logs

A. Mazak, S. Wolny and M. Wimmer
DOI: 10.1109/ETFA.2019.8869502

14 Model-driven Runtime State Identification

S. Wolny, A. Mazak, M. Wimmer and C. Huemer
Corpus ID: 220055432

15 From AutomationML to AutomationQL:

A By-Example Query Language for CPPS Engineering Models

A. Mazak and M. Wimmer
DOI: 10.1109/COASE.2018.8560448
1 Introduction

1.1 Motivation

Today, most of the systems we interact with are linked to a globally networked infrastructure in which products, processes, and resources interact with embedded hardware and software far beyond the scope of a single application \[24\]. For instance, in the automated manufacturing engineering domain, the Internet of Things (IoT) creates cyber-physical networks transforming the traditional production to a so-called “smart production” \[18\] \[42\] \[85\]. Cyber Physical Systems (CPS) enable the convergence of the physical world and the virtual world in a higher level environment \[71\]. In CPS, the physical environment is populated by highly interwoven communicating objects, such as networked controllers, sensors, actuators, and other smart devices \[129\]. Therefore, flexible monitoring and control approaches are needed to adapt the systems’ behavior to ever-changing requirements and tasks, unexpected conditions, as well as structural transformations \[81\]. CPS components are capable of autonomously interacting with each other and with the environment itself. As a result, both, the volume and the level of detail of data generated are highly increasing. Due to this ever growing complexity also resulting from shorter innovation cycles, rapidly changing customer needs, etc., the evolutionary aspect of engineering artifacts that change over time is becoming more and more important \[82\] \[49\] \[133\]. This requires real-time communication, seamless data exchange as well as data stream analysis \[131\] \[134\] \[135\].

In order to deal with such software- and data-intensive systems, modeling is considered as a promising technique to understand and simplify reality through abstraction, and thus, models are in the center of the engineering process \[13\]. In a Model-Driven Engineering (MDE) approach, models represent the most important artifacts throughout all, most often interdisciplinary activities during an engineering process \[13\]. There are many kinds of models such as simulation models, prescriptive models, descriptive models, statistical models, planning models, etc. From an MDE perspective, when modeling the digital side of a system (such as the controller of a machine), the software is developed and implemented in a model-driven way \[23\]. This means that throughout this engineering process, models are the main “driver” and not solely used, e.g., for documentation purposes \[13\].

On the one hand, a model serves as an abstraction for a specific purpose, as a kind of “blueprint” concentrating on a system’s structure as well as desired behavior exe-
cuted by actuators based on controller commands. Accordingly, such design models represent early snapshots of the system. On the other hand, there are so-called runtime models providing real abstractions of systems during operation basing on the data gathered through sensors. There is however a discrepancy between models of design time (development phase) and models of runtime (operation phase) [93, 43]. For reducing the aforementioned complexity in CPS as well as for monitoring systems over time, an a-priori description of a system at hand is not enough [132]. It requires a connection from such an initial model to its runtime counterpart. Accordingly, models should be held “in-the-loop” from design to operation in order to be fully integrated in the life cycle of a system. Thus, the temporal aspect of models has to be taken into account in MDE.

The remainder of the introduction chapter of this cumulative habilitation thesis is structured as follows: In Section 1.2, we discuss the foundational concepts that provide the basis for our research work. We briefly introduce MDE principles followed by an overview on the important subtopics metamodeling, modeling, languages as well as temporal languages in general. Section 1.3 elaborates on the need for Temporal Model-Driven Systems Engineering. The challenges when developing and implementing this approach are discussed in Section 1.4. These challenges relate to data connectivity and integration issues, the handling of temporal models, and semantic-based data analytics. Section 1.5 reflects the core of the habilitation thesis by elaborating on our contributions which are divided into three main parts: (i) Model-Driven Connectivity, (ii) Temporal Model Management, and (iii) Data-Driven Model Analytics. Since these contributions would not have been possible without appropriate funding, we also outline the research projects enabling our research works. We provide a mapping between the identified challenges and presented contributions. Furthermore, we show which papers contribute to solve these challenges and discuss each contribution in more detail. A summary in Section 1.6 and an outlook in Section 1.7 conclude the introduction chapter.

1.2 Background

1.2.1 Model-Driven Engineering

In Model-Driven Engineering (MDE), the abstraction power of models is used to tackle the complexity of systems [117, 72, 23]. From an abstracted point of view, MDE follows the principle “everything is a model” [14]. This means MDE supports system- and software engineers by providing formal models, like a tool box, to achieve simplicity, generality, and integration when modeling a system.

Historically, MDE has been mainly applied in Software Engineering [14, 23], but in recent years, MDE has also emerged in Systems Engineering [13, 146], such as in the in-
1.2 Background

Industrial automation domain \[64,136,119,12\], or in the Architecture, Engineering, Construction (AEC) industry \[99,110\]. Furthermore, MDE concepts have been extended by runtime aspects \[93,11,37,9,59,73\]. In MDE, models are the central artifact used as a main “driver” throughout the development process, finally leading to an automated generation of systems \[40\]. Therefore, MDE models are considered as first class citizens \[15\]. Well distinguished from MDE is Model-Based Systems Engineering (MBSE) which is considered as a “softer” version of MDE. According to the International Council on Systems Engineering (INCOSE) MBSE is defined as “the formalized application of modeling to support system requirements, design, analysis, verification and validation activities beginning in the conceptual design phase and continuing throughout development and later life cycle phases” \[67\].

In the following, we give a short overview on the difference between Model-Based Engineering (MBE) and MDE according to \[23\]. In MBE-processes models play an important role although they are not necessarily the key artifacts of the development. This means that they do not “drive” the process like in MDE. In MBE, models are created as a kind of “blueprint” directly handed out to the programmers for having a guideline for manually implemented code. There is no need for an automatic generation by code generators and there is no explicit definition of any platform-specific model. In a nutshell, models have an important role, but are not the central artifacts as in MDE processes, and therefore, MBE is defined as a superset of MDE \[12,23\].

In the early phase of system development in MDE so-called prescriptive models are used to create and describe the scope of interest in a certain granularity of detail \[65\]. Thus, by definition, a model never describes reality in its entirety, rather it describes a scope of reality for a certain purpose in a given context \[23\]. Mayr et al. \[88\] critically note that models are mainly used as prescriptive documents only, and therefore, aim for a model-centered architecture paradigm. Their intention is to keep models, and in general, any developed artifact synchronized in all phases of development as well as in the operation phase. Therefore, in the later phases of the system life cycle so-called descriptive models may be employed to better understand how the system is actually realized and how it is operating in a certain environment \[65\]. Compared to prescriptive models, such descriptive models are only marginally explored in the field of MDE, and if used at all, they are created manually \[93\].

The implementation phase deals with the mapping of such prescriptive models to executable systems and consists of three levels \[23\]: (i) the modeling level where the models are defined, (ii) the realization level where the solutions are implemented through artifacts that are then employed in the running system, and (iii) the automation level where mappings from the modeling to the realization phase are defined. Those mappings enable to automatically transform model elements to code statements which can be executed on a platform \[23\]. For this step, MDE provides model transformations as one of its key techniques, which are Model-to-Model (M2M) and Model-to-Text (M2T) trans-
formations [14]. Additionally, other MDE techniques such as model validation, model verification, model simulation, or model execution enable the automation of engineering process steps and support the traceability of engineering artifacts in general [23].

1.2.2 Metamodelling and Modeling Languages

The aforementioned levels are currently only used for downstream processes. The possibility of considering upstream processes, by gathering value streams of running systems, is rather neglected in MDE [93]. This means that even if systems are described by means of modeling languages and code generators are used to transform model elements to corresponding code statements, the execution of those statements is typically not represented in the metamodel.

In general, modeling languages are defined by metamodels. Metamodels are used to describe the abstract syntax of modeling languages. Based on this, the concrete syntax and semantics are defined to model a domain of interest (e.g., software controller of a machine) [23]. This means that each model created by using a modeling language is an instance of its metamodel, and thus, has to conform to it [15]. Additionally, whenever needed, metamodels can be adapted or extended, e.g., to tackle certain problem spaces of a domain. In addition, metamodelling enables (i) to compare models and reason about differences between them, (ii) to align models to create an integrated representation of a system, and (iii) to translate to other formalisms such as for code generation.

The Unified Modeling Language (UML) [106] is one of the best known modeling languages based on the Meta Object Facility (MOF) [109] standard. The advantages of UML are platform independence as well as adaption and extension capabilities for users to meet their own requirements for modeling specific interests. UML offers a wide range of views and different types of diagrams to represent the structure and behavior of a system [121]. One example of a metamodelling language that is based on a core subset of UML and MOF is Ecore from the Eclipse Modeling Framework (EMF) [45]. Since, Ecore supports the key concepts of using models as input to development and integration tools, it is one of the most widely used languages for code generation and model serialization for data interchange. We use UML and an extended subset of it known as Systems Modeling Language (SysML) [125] as well as Ecore for most of our contributions as presented in Section 1.5. Like UML, SysML is an Object Management Group (OMG) [105] standard, a so-called general-purpose modeling language, providing a graphical modeling language for describing complex systems by considering software as well as hardware parts. It should be briefly mentioned, that, additionally to these general modeling languages, there are Domain Specific Languages (DSLs) specialized for a specific application domain, or a specific purpose, with their own notations [23].

While in the early phases model-driven approaches are frequently used to design systems, in the later phases data-driven approaches are used, for instance, to reason on dif-
ferent key performance indicators of systems when operating in an environment [98]. This immediately poses the question how operational data can be mapped back to design models to evaluate existing designs and to reason about future re-designs. Thus, the combination of model-driven and data-driven approaches is required. Otherwise it would be difficult to prove whether the design model corresponds to its runtime counterpart. Even though the identification of discrepancies is not straight forward, engineers in industry would benefit if they could treat runtime data in the same way like standard UML, or SysML models.

With the emergence of CPS and other sophisticated runtime monitoring infrastructures, time-series databases [7] are frequently applied to store historical data of systems as well as to provide powerful analysis by dedicated query languages. There is abundant research on temporal extensions for modeling languages to specify the temporal characteristics of the system data (e.g., consider the survey of [62]), but not regarding the temporal dimensions of models themselves. Further works advance these first attempts by extending also the query languages with temporal properties, mainly to enable the validation and verification of temporal properties of the data.

Temporal OCL (TOCL) [151] and Temporal UML [28] are two examples of OCL extensions for the evaluation of temporal constraints. Temporal extensions have also been applied to specific types of systems (e.g., adaptive systems [103]) and DSLs (e.g., timed Petri nets [10, 78]). Even TOCL, which can be seen as a generic language, can also be used as a component in other DSLs as described in [100]. In this line, Bousse et al. [22] discuss and apply a pattern to extend modeling languages by events, traces, and further runtime concepts. The extention is used to represent the state of a model’s execution. Futhermore, it provides verification support by applying TOCL for defining properties that are, together with the models, mapped to formal domains. Efficiency of these types of temporal inspection queries are also in the focus of [54] as well as [55].

All these approaches are mostly oriented towards the retrieval of specific past states of the model/data, elaborating on the concepts of valid time and transaction time of (bi)temporal models. Instead, we explicitly focus on the support for complete time-series storage and analysis, which opens the door to more powerful and rich possibilities, like the computation of different key performance indicators for models as part of design exploration or simulation scenarios.

1.3 On the Need for Temporal Model-Driven Systems Engineering

Today, evolving models are stored in model repositories, which often rely on existing versioning systems or standard database technologies [16]. These approaches are sufficient for hosting different model versions of an evolving model by storing each version
separately as self-contained model instance. Those different versions of a model allow to reason about evolution concerns. However, the time dimension is not explicitly targeted on the model element level, and therefore, is not accessible. In order to support systems engineering processes where models are not only used in the system design phase, but also, in the operation phase, a more explicit representation of time is needed in order to reflect and reason about temporal aspects, such as element or state changes over time. Thus, the explicit consideration of temporal aspects on the level of model elements is essential from design to operation and backwards by combining downstream information from the MDE-process with upstream information gathered at runtime. Thus, a novel approach is needed that provides models on time-series databases for enabling runtime queries and for enriching models with historical data from the operation phases. This additionally enables a so-called data-driven model analytics \cite{98} in order to analyze the actual runtime behavior of a system compared to the one initially designed. Thereby, the challenge is not only to bundle sensor value streams (e.g., from IoT networks) and aggregate them to a higher logical state level for process-oriented viewpoints, but also to consider uncertainties of the realization precision of sensor measurements during long-running operations. This allows to build a “vertical bridge” from the operation technology layer to the IT layer, where process views are integrated and model-based monitoring as well as analytics may be performed.

1.4 Challenges of Temporal Model-Driven Systems Engineering

In this section, we give a brief summary of research works related to the approaches presented in this cumulative thesis and discuss challenges in the scientific communities by specifically focusing on (i) heterogeneity management, (ii) temporal models, (iii) runtime models, and (iv) data analytics. In order to truly integrate models in a system life cycle, dedicated foundational as well as applied research efforts are needed.

1.4.1 Heterogeneity Management

The full integration of models in the system life cycle requires a continuous acquisition of real-time data (e.g., machine data, sensor data, event streams) from various distributed devices. Thereby, the data flow of those sources goes hand in hand with the translation among different data models with different syntax and semantics \cite{25}. Another important challenge when linking data to models is the integration of data available in different formats (XML, native database formats, comma separated file formats, etc.). For this purpose an alignment and mapping of different entities needs to be accomplished. Each system is usually described by a multitude of models. The differences are a result of producing models of different granularity with continuous re-
1.4 Challenges of Temporal Model-Driven Systems Engineering

finement during the engineering process. Also, the application of different tools in this process indicates differences when modeling a domain of interest [120]. Evidently, the information in all these models is partially overlapping and partially disjoint. It is essential to carry information of existing models forward to activities creating new models. Thus, model transformations bridging syntactical and semantic issues are needed.

**Challenge I: Connectivity and Integration**

The challenge of providing connectivity and integration can be divided into two subparts: (i) technical integration by means of appropriate interfaces, and (ii) semantic integration [104]. The latter one has to consider data and object models for storing and dealing with historical data, parameters and configurations. There are several standards used for horizontal as well as vertical integration on different layers, especially in the manufacturing domain. Furthermore, it requires approaches at the interface between technical and semantic integration, such as ISO 15926 which is used to store life cycle information as discussed in [80], to mention only one example. Depending on the application context, there is a plethora of industry standards that have to be investigated for the combination and revamping of information from heterogeneous sources into new homogenized representations.

1.4.2 Runtime and Temporal Models

Most approaches for runtime modeling aim for bridging the gap between design time modeling and runtime modeling to enable runtime analysis. Blair et al. [19] show the importance of supporting runtime adaptations to extend the use of MDE. These so-called *Models@run.time* provide a formal basis for dynamic adaptations, analysis, and predictions of a system when operating for supporting dynamic adaptation [102]. Different stakeholders can use those models in various ways for runtime monitoring such as dynamic state monitoring. Hartmann et al. [63] combine the concept of runtime models with reactive programming and peer-to-peer distribution. Reactive programming aims for enabling support for interactive applications, which react on events by focusing on real-time data streams. For this purpose a typical publish/subscribe pattern, well known as observer pattern in software engineering [53], is used. Khare et al. [76] show the application of such an approach in the IoT domain.

Hartmann et al. [63] define runtime models as a stream of model chunks, as it is common in reactive programming. The models are continuously updated during runtime, therefore they grow indefinitely. With their interpretation that every chunk has the data of one model element, they process them piecewise without looking at the total size. In order to prevent the exchange of full runtime models, peer-to-peer distribution is used between nodes to exchange model chunks. In addition, automatic reloading mechanisms are used to respond to events. As the models are distributed, operations
like transformations have to be adapted. For this purpose transformations on streams 
as proposed by Cuadrado and de Lara [38] as well as Dávid et al. [39] can be used.

In order to successfully implement such runtime models and to bridge the gap be-
tween design-time modeling and runtime modeling, the temporal aspect of data is 
necessary to be included in the investigations. Initial work was done in the field of 
temporal relational databases, as presented in [26], followed by approaches present-
ing dedicated mappings of design models to different data base technologies on the 
basis of different data paradigms as discussed in [58]. The research works in this con-
text were continued and extended in the area of temporal data warehouses [60], and 
have mainly resulted in the explicit support of temporal SQL in many existing rela-
tional database systems. Especially, multi-version object-oriented databases allow for 
revisions to model evolution in time and variants enabling parallel ways of develop-
ment [29, 115]. In addition, there are research works in the domain of CAD engineering 
tools and accompanying data stores [115].

These approaches have in common that the temporal aspect is solely a record with a 
timestamp. Additionally, the data models are residing “behind the walls” of reposi-
tories without an appropriate connection to the running system. To keep data synchro-
nized and consistent between the system’s design and runtime phase, an appropriate 
temporal framework is needed. Such a framework must address the hosting of data of 
runtime monitoring as well as the handling of temporal models that go beyond repre-
senting and processing the current state of a system and its components [61, 150, 16]. 
This enables the shift from monolithic models to evolutionary models populated by 
timing aspects, where the focus is not only on the current state to steer the system but 
also on the history of changes.

**Challenge II: Handling Temporal Models**

To tackle the limitation and open issues described above, it is essential to focus not only 
on providing a model infrastructure to instantiate models but on the history of changes 
of those models and their components (e.g., state changes, value changes) during the 
system’s operation. For this purpose a specific temporal model framework is required 
that enables an automatic monitoring of systems by using temporal models. To fur-
ther close the gap between design time modeling activities and runtime monitoring 
activities [59], an explicit mapping of design models to time series databases that can 
be considered as a special type of temporal databases [20, 116] is needed. This man-
dates a dedicated profile for extending metamodels with an appropriate annotation 
mechanism to drive and optimize the generation of model-based time-series database 
connectors. In addition, an appropriate mapper is needed that allows to inject data to 
time-series databases from model changes as well as to extract data from the time-series 
databases by model-based queries, e.g., in OCL [27], for efficient monitoring purposes.
1.4 Challenges of Temporal Model-Driven Systems Engineering

1.4.3 Data Analytics

The term data analytics subsumes techniques examining big data sets to uncover hidden patterns, unknown correlations, and other information that is used to make better decisions. The steps that data passes through in most applications are the following: acquiring, cleansing, storing, exploring, learning, and predicting [111]. Mostly, the acquisition of information that derive from a big amount of data gathered from heterogeneous sources is challenging. Since, there are very large data volumes (Volume), data arrives very fast in form of data streams (Velocity), and data has varying and complex formats, types, and meanings (Variety) [5, 79]. Later on, further characteristics of big data, that are not discussed in detail here, were identified: value, veracity, viscosity, variability, volatility, viability, and validity [25]. These characteristics of big data projects put high requirements on data storage (e.g., NoSQL, RDBMS), data processing (e.g., batch, incremental, interactive), orchestration (scheduling, provisioning), and interfaces for offering access points (e.g., SQL, script, graphical) to mention only four of six pillars of the big data analytic ecosystem, as introduced in [74].

A multitude of methods and techniques have emerged to support data analytics, such as: data mining [50], the CRISP-DM process [143], pattern recognition and machine learning [17], anomaly detection (e.g., in [30]), predictive as well as prescriptive analytics (e.g., [123, 111]), and process mining [127]. At the end the aim is to recognize meaningful data models within big data, e.g., to recover existing patterns and to discover new ones [41].

Challenge III: Semantic Data Analytics

The model-driven perspective describes how an intended system should work. The data-driven perspective focuses on how data logs can be taken from a system during operation to reason about the actual realization of the system. While the former perspective is lacking concepts to define runtime data such as time-series, the latter one has to correctly interpret the collected data logs. Often, data is stored in various text-based formats (e.g., XML, comma separated file formats), which are difficult to read and process, since the stored information is not obvious. Thus, it is not possible to easily trace runtime processes in the system and between system components. This requires a scalable reverse-engineering approach to provide an object-oriented view on this data and to enable the reverse engineering of runtime information into design models. The challenge is to overcome the gap between both perspectives by (i) monitoring meaningful data from operation, (ii) aligning data logs with the design model for providing a semantic anchoring of data, and (iii) providing meaningful analytics to reason about improvements or fulfillment of given requirements.
1.5 Contributions to Temporal Model-Driven Systems Engineering

This section presents in detail how the papers (Papers 1 to 14) part of this cumulative habilitation thesis address the challenges discussed in Section 1.4. It elaborates on solutions for considering interoperability aspects as well as evolutionary aspects of engineering artifacts by adding a temporal dimension to models and model elements.

1.5.1 Overview

Figure 1.1: Unifying Framework for Temporal Model-Driven Systems Engineering.

The contributions are located in three main research areas: (i) Model-Driven Connectivity, (ii) Temporal Model Management, and (iii) Data-Driven Model Analytics. Figure 1.1 illustrates how these contributions are arranged. In particular, the focus is on how to connect runtime environments (i.e., heterogeneous data sources) to a temporal model framework to extract so-called operation models from the Data Layer, connect them to design models at the Model Layer, and to establish (logical) model analytics at the Analytics Layer. For this purpose an appropriate language extension on the
1.5 Contributions to Temporal Model-Driven Systems Engineering

Language Layer is required based on the operational semantics of the employed language. Thus, the contributions are as follows:

(i) *Model-Driven Connectivity* deals with the combination and revamping of information from heterogeneous sources into new representations and provides a model-driven approach to accumulate various engineering views from embedded systems over network technologies. Thus, it tackles the integration of heterogeneous tools from various engineering domains (i.e., manufacturing, Architecture, Engineering, Construction (AEC) industry) through mappings realized as model transformations of underlying industry standards.

(ii) *Temporal Model Management* provides the capability to handle and store runtime as well as historical data by a unifying framework linking this data to temporal models, and additionally, providing query capabilities. In particular, we enrich models with a temporal component to shift them from static to evolutionary artifacts on the basis of the data gathered in the operation phase of a system. This enables an explicit representation of time on the model level as well as on the model element level.

(iii) *Data-Driven Model Analytics* investigates on the issue of model element changes over time. We apply model-driven approaches to reason about a system’s behavior and model element changes based on the descriptive (operational) models obtained from runtime.

These contributions were evaluated by either observational, analytical, experimental, or descriptive methods. For this purpose we built up three lab-sized automation system environments with an increasing complexity: a traffic light system, a self-driving car, and a five axes grip-arm robot. Additionally, we employed test-beds from research partners of the Otto-v.-Guericke University Magdeburg (e.g., as presented in Paper 11) and the CTU’s Czech Institute of Informatics (e.g., as presented in Paper 5).

The research work on the papers collected in this thesis has been conducted in the context of the following research projects where the applicant has been acting as a scientific leader and/or principal investigator:

- InteGra 4.0 — Horizontal and Vertical Interface Integration 4.0 an Exploratory Study, FFG ICT of the Future, Austrian Research Promotion Agency (FFG) and the Federal Ministry of Science, Research and Economy (BMWF), [849944], 2015–2016, Principal Investigator;
- DigiTrans 4.0 — Innovationslehrgang zur Digitalen Transformation in der Produktentwicklung und Produktion, FFG Innovationslehrgänge, Austrian Research Promotion Agency (FFG) and Austrian Federal Ministry for Digital and Economic Affairs (BMDW), [854157], 2016–2018, Principal Investigator;
• CDL-MINT — Christian Doppler Laboratory for Model-Integrated Smart Production, Austrian Federal Ministry for Digital and Economic Affairs (BMDW) and the National Foundation for Research, Technology and Development (CDG), 2017–2020, Scientific Lead of the Module Reactive Model Repositories;


Figure 1.2: Mapping of Contributions to Challenges.

Figure 1.2 gives an overview of the contributions, which we map to the identified challenges. Furthermore, we specify the papers including the contributions to solve the challenges. In the following, we present the list of these papers:

**Contribution 1: Model-Driven Connectivity**


**Contribution 2: Temporal Model Management**


**Contribution 3: Data-Driven Model Analytics**


In the following, each of the listed contributions is presented in more detail. We explain how each challenge is addressed according to the type of the contribution. Furthermore, we present the evaluation of each of the contributions.

1.5.2 Contribution 1: Model-Driven Connectivity

The contributions of the presented papers in this section deal with the combination and revamping of data from heterogeneous sources into new and harmonized representations for providing meaningful insights. Furthermore, our approaches yield towards a nearly seamless exchange of relevant information, both vertical (from the data sources to the data layer downwards and upwards, cf. Figure 1.1) and horizontal (from one system to another at the same layer). After resolving certain forms of heterogeneity in the handling and integration of data, the data is processed within distributed operational models for further elaboration.

Paper 1: From business functions to control functions: Transforming REA to ISA-95 [90]

In the context of Industry 4.0, a seamless information exchange between information systems on the same layer (i.e., horizontal integration) and between information systems on different layers (i.e., vertical integration) is a key issue. In 2013, the German working committee for Industry 4.0 pointed out that production systems are to be linked vertically to business processes within decentralized production sites and enterprises, and that they are to be distributed horizontally among suppliers, distributors, and companies. In order to meet these requirements, we aim for an integrated modeling framework spanning over the horizontal layer (value networks) and the vertical one (production chains). Thereby, MDE acts as an enabler for managing such an interface integration framework. This means that we do not start from scratch by defining an own all-encompassing modeling language, rather we build up on existing well defined standardized ones.

For vertical integration of information flows within the company, we consider the concepts and models of the standard ISA-95 (ANSI/ISA-95; IEC 62264) [69]. This interna-
tional standard describes information flows between Enterprise Resource Planning Systems (ERP) on the enterprise level and the Manufacturing Execution Systems (MES) on the control level. The interface for the horizontal integration of information flows of different business partners is described by the concepts of the Resource-Event-Agent business ontology (REA) (ISO 15944-4) [1]. In the business domain, REA is used to identify the value-adding activities of a company. For realizing integration by means of value networks, we transform REA concepts to ISA-95 models (e.g., material model). This approach is independent of any software solution. This means that companies may use different ERP and MES systems and still collaborate with each other.

**Type of Contribution:** We present a meta model (cf. Figure 1.1, at the Language Layer) for REA and for ISA-95 to align the concepts of both. For this purpose we extend the resource concepts of REA by similar concepts from ISA-95 for providing specialized concepts of resources such as equipment, physical asset, and material. These extensions concern the REA type level. Most importantly, we develop dedicated transformation rules for an automatic mapping to transform a REA model into a ISA-95 one. In particular, we map REA duality models to ISA-95 operation segments. This enables converting information about input and output of business functions to control functions.

**Validation of Contribution:** In a first step, we implemented the proposed REA extensions into our REA-DSL tool, which we developed in previous research work [89]. In a second step, we added transformation rules. In this initial paper, we hard coded these rules. In subsequent contributions (see Paper 2 and Paper 4), we worked on an automated generation of these rules. We demonstrated the technical feasibility by mapping from our REA DSL to B2MML (the XML equivalent of ISA-95). The syntactical correctness of the transformation was checked by the proof of valid B2MML instances.

**Paper 2: AutomationML, ISA-95 and Others: Rendezvous in the OPC UA Universe** [139]

Based on our lessons learned of Paper 1, we extend the previously presented approach by exploiting two additional standards, the Automation Markup Language (AutomationML) [34] and the OPC Unified Architecture (UA) [52]. AutomationML consists of three parts: (i) CAEX, (ii) COLLADA, and (iii) PLCOpen. CAEX (Computer Aided Engineering Exchange) is a data format that has been defined in the scope of IEC 62424 [32]. CAEX is based on XML and enables the metamodeling and modeling of, e.g., the hierarchical architecture of a plant, including involved machines and controllers and their physical and logical connections. AutomationML defines an abstract interface class ExternalDataConnector which is used to reference external documents and elements therein. Two use cases of this external data connector have been defined in separate whitepapers: (i) COLLADAInterface specifies how external COLLADA3 documents are referenced [36] and (ii) PLCopenXMLInterface defines how PLCopen4XML documents [2] (which are based on IEC 61131-3 [33]) can be referenced from AutomationML documents [35].
This referencing mechanism plays an important role in the analysis of the presented contribution.

OPC UA is a commonly used communication standard in industrial settings. It is a versatile platform for hosting information from a large variety of domains [31]. Those domains provide only partially overlapping information, since there are different views on a specific entity or different levels of detail needed for describing a specific interest, etc. Emerging from a multi-disciplinary engineering process, these different views can stem from various tools that have been used to deal with that entity, or from different stages in an engineering process, e.g., from requirements engineering over system design and implementation to operations.

The problem that needs to be addressed is that domain-specific information is kept out of the OPC UA standard as strictly as possible [31]. Instead, companion specifications are used to extend OPC UA by introducing meaning (semantics) depending on the domain of interest. Furthermore, numerous engineering tools are used in the design (and runtime) of automated productions systems. The high number of combinations of engineering tools and companion specifications leads to a great interoperability challenge. We argue that a concise but expressive set of OPC UA reference types that allow the persistent instantiation of additional knowledge with respect to relations between OPC UA nodes may help with the semantic alignment of such diversified entities.

**Type of Contribution:** We provide an OPC UA information model for explicitly linking heterogeneous data, generated from different source domains. This information model provides a concise but expressive set of OPC UA reference types: **RepresentDifferentViews**, **HasRefinement**, **HasVerification**, **HasImplementation** that allow the explication of relations between engineering artifacts. Instances of these reference types are meant to be used between OPC UA nodes of different domains. However, they can also be applied in the context of a single domain, if this domain does not provide corresponding reference types that might be of value. These reference types can be sub-classed to describe more specialized inter-model relations, if required.

**Validation of Contribution:** We created a set of domain models based on AutomationML, ISA-95, and MTConnect [51] to evaluate our set of OPC UA reference types. For each of these models there exists a mapping to OPC UA, or they are natively designed in OPC UA. As evaluation example, we considered a milling device. We represented this device named *MyMilling* as three different components at an OPC UA server: (i) as an AutomationML internal element, (ii) as an ISA-95 physical asset, and (iii) as a MTConnect device. Using the **RepresentDifferentView** reference type it becomes clear that the different OPC UA nodes are all describing the same physical entity, a milling device, but from different domain specific perspectives. A similar example was given with the entity *MyWorker*. In AutomationML it is a proxy element (usually personnel is not modeled in AutomationML), but modeled in more detail in ISA-95 by the personnel model. This refinement was expressed using a **HasRefinement** relation. The
validation of the type set showed that there are use cases in which such explicit links make sense, e.g., when the domain model does not provide a facility to describe the relations as discussed.

**Paper 3: Leveraging integration facades for model-based tool interoperability** [110]

In the domain of Architecture, Engineering and Construction (AEC) industries, the volume of data to be exchanged among various stakeholders (e.g., building developers, energy suppliers, architects, structural engineers, building physicists, etc.) is rapidly increasing in daily business. Each of these stakeholders possesses specific domain knowledge and has a specific view on the building project. These different perspectives may cause different forms of heterogeneity in the definition and handling of data that hinders an interference-free communication. It is still common practice that IT systems exchange information through extensive interfaces, but can only utilize specific pieces of that information. The situation is further worsened by the problem that many different interfaces introduce dependencies whose management can become complex and hard to achieve. This leads to a drastically rising complexity of the system. In this paper, we focus again on multi-disciplinary engineering processes. We discuss the lack of interoperability among different domain-specific engineering tools, and heterogeneity issues resulting from different perspectives of stakeholders on the same entity as well as on different information granularity needed in various project phases. In particular, we transfer the lessons learned of Paper 2 to the application field of the AEC industry and extend that previous research work.

For this purpose we explicitly formulate challenges to work on when implementing a road map towards full semantic and pragmatic integration that need to be tackled in any data exchange process, namely exposing semantics, exposing functionality, staying up-to-date, and making pragmatics explicit. These challenges address a seamless integration considering the semantics and pragmatics of a single application. Nevertheless, full interoperability requires an unbroken communication network involving multiple applications. Additionally, the data exchange within or across different phases of a building project is challenging due to different exchange standards in use. Therefore, Building Information Modeling (BIM) has become more and more established in recent years [124]. The main motivation of BIM is accommodating heterogeneous nature of the AEC industries and involved domains, and providing seamless data flows within any building or infrastructure project [21].

**Type of Contribution:** Interoperability in BIM requires integration along, both, the semantic and the pragmatic dimensions. In this contribution, we present and evaluate a modeling framework capable of working with multiple semantic type systems inhabiting the same syntax in the context of multiple applications. In addition, the framework
provides formal methods for modeling the pragmatic aspects of a data exchange and converting them from an implicit assumption into an explicit formal specification. It enables the integration of semantics at runtime and of pragmatics at design time. This combined model of semantics and pragmatics provides a so-called integration facade that enables transparency and traceability during interoperability testing.

**Validation of Contribution:** Use Case 1: an application simulating particle motion under gravity. This software simulates the motion of a swarm of particles, each with an initial mass and velocity, under the influence of gravity. The results are represented by an animation on a two-dimensional canvas and as a table containing mass, positions and velocities. Use Case 2: an application simulating the propagation of sound waves. This software simulates the propagation of sound waves generated by user-defined sound emitting point or line sources. The resulting interference pattern is calculated numerically over a discrete grid and is displayed as an animation on a canvas in one, two, or three dimensions. With these two use cases we validated the overcome of the addressed Challenge 1: exposing semantics and Challenge 2: exposing functionality. For Use Case 3, we used a spreadsheet for simulating the thermal behavior of a single space. Thereby, we calculated the temperature, humidity, and CO$_2$ concentration in a single enclosed space over the period of one week during a heat wave in summer. This case demonstrated the steps involved in addressing Challenge 4: translating between different semantics.

In all of these three use cases, we made adaptions in the source code and evaluated both the adaption itself and the implementation of the presented domain-specific data exchange workflows. The results show that the prototype of our modeling framework for integration facades provides full integration for any data model, i.e., a semantic and pragmatic consensus. This independence of concerns, allows the semantics to be modelled independently of the notation or representation of information in any available tool, which makes our approach universally applicable to data exchange processes. In addition, we can model pragmatics, both, along the semantic and the representational dimensions. This makes a full integration of semantics and pragmatics feasible into an integration facade, which is a prerequisite for producing a so-called “single source of truth” as envisioned for Big Open BIM.

**Paper 4: A View on Model-Driven Vertical Integration: Alignment of Production Facility Models and Business Models**

With this paper, we continue our work of Paper 1. In this Paper 4, the application focus is also on Automated Production Systems (aPS) where modern IT systems are required at all levels of the automation hierarchy: from business-related software at the corporate management level, down to the programmable logic controllers at the field level. For a well-designed coupling of systems that are located at different levels, it is necessary to identify, define, and implement clear data conversion mechanisms. This
endeavor is also known as vertical integration as already explained in Paper 1 and is further elaborated in Paper 5.

In a vertical integration scenario, IT systems of different vendors might be in use and proprietary interfaces need to be defined in order to exchange relevant information from one system to another, an issue we already discussed and worked on in Paper 3.

In this paper, we present an MDE approach for the co-evolution of models, residing on different levels of the automation hierarchy, based on a generic alignment of corresponding metamodels through appropriate model transformations (cf. Section 1.2.2).

**Type of Contribution**: The contribution is an MDE-based architectural metamodel for vertical integration of IT systems. We integrate parts of AutomationML, IEC 62264-2, and REA (cf. Paper 1) through model transformations techniques, more specific by M2M-transformations. The metamodels of these three industry standards are used for the representation of hierarchy level-specific system properties and the alignment of key concepts in order to provide bridging functions for the transformation between different IT systems. We provide explicit mappings between model elements, and therefore, do not rely solely on the metamodel level (cf. Figure 1.1 at the Language Layer).

The approach enables that (i) changes in one system are automatically propagated to other systems, if possible, and (ii) the overall architectural model is not modeled explicitly, but to be inferred from multiple domain models. We employ the Epsilon Object Language (EOL) [46] for querying model states, Epsilon Transformation Language (ETL) [47] for M2M-transformations, and EVL [48] for the validation of models.

While the given technique does not provide a single point of intervention when it comes to changes in the models, it facilitates the creation of stub models and provides means for cross-model validation. The main contribution of this paper is the model-driven propagation of basic model elements and changes of model elements between models of different hierarchy levels.

**Validation of Contribution**: We validated the approach by an in-depth domain analysis using an application scenario of a fictitious company which we named Glulam Ltd., specialized in the production of glued laminated timber. The idea was born from a visit in a real company of the woodworking sector in the course of the InteGra 4.0 project. Even though, we evaluated the application of our approach on a very specific fictitious company, the approach itself and most of the implementation are company as well as domain agnostic. In the evaluation results, we refrained from presenting details about the transformation between AutomationML and ISA-95 models, as the alignment between these two metamodels was demonstrated in more detail in our research work presented in [138] (a paper which is not part of this cumulative habilitation thesis). This means that corresponding transformation rules can be inferred from the mappings defined in that previous paper. The results show that the developed metamodel provides a semantically enriched revision of the XML schema-based original definition of the CAEX format, and so of AutomationML.
The evaluation figures out that the benefit of this approach is that a common understanding of concepts from different domains is accomplished by relating metamodel elements to each other. This approach is agnostic to the kind of business a company is involved. Specific implementations could consider industry-related information in order to better acknowledge peculiarities and conventions.

**Paper 5: Flexible Production Systems: Automated Generation of Operations Plans Based on ISA-95 and PDDL [140]**

Since, the standard IEC 62264 (see Paper 1 and 4) provides a conceptual model for the representation of manufacturing operation management information, in this paper we explore how this information could be exploited for an automated generation of production plans. For this purpose we use the Planning Domain Definition Language (PDDL) [57] as encoding format. PDDL provides a standardized and object-oriented way of specifying planning domains and concrete planning problems.

The idea behind the presented approach is that IEC62264 (as already showcased in Paper 1) can be used to model (i) the machinery of a production system (the equipment), (ii) the material that is being consumed and produced, (iii) the production processes that are available (the so-called process segments), as well as (iv) the relations that arbitrary resources can have to each other. Our approach establishes a conversion rule from IEC 62264 information to PDDL information. In essence, the IEC 62264 process segments are translated into PDDL actions that could be applied by a planning solver in order to progress from an initial state to a goal state. In our presented use case, we automatically compute the sequence of actions that will be necessary to prepare just-in-time-delivery of raw material for a production process that is to be executed later, by ordering a sequence of shuttles in a monorail intra-logistic transportation system.

**Type of Contribution:** The main contribution is the transformation of IEC 62264 information into a valid PDDL representation. For this purpose we develop a concrete metamodel and appropriate M2T-transformations for PDDL. This provides the transformation of pure IEC 62264 metamodel information to PDDL domain description fragments which we examine by the example of equipment information. By employing this metamodel, classes are made available as PDDL types and metamodel relations are transformed to PDDL predicates with a corresponding name as well as properly typed parameters. The PDDL elements generated in this way serve as the backbone for the remaining elements that are inferred from IEC 62264 model information. For instance, (i) all classes (e.g., equipment classes) are converted into constants, (ii) process segments are converted into actions and their segment requirements are added thereto as parameters, and (iii) all kinds of instances (e.g., equipment) are converted into objects of a corresponding problem definition file.

The so computed production plan is then written back into the production system.
model in terms of an operations definition, where each PDDL action call is converted into an operations segment. If no plan can be found, no such operations definition is created, and instead, the user is notified that no plan could be found for the given production problem. The corresponding engineer is then able to explore whether the product is faulty, or the production system is simply not capable of producing the selected kind of product.

Validation of Contribution: We applied our research concepts on the basis of a lab-sized Industry 4.0 testbed at the CTU’s Czech Institute of Informatics, Robotics and Cybernetics (CIIRC) [66]. This testbed comprises four robot cells which are connected by a monorail-based intra-logistics system featuring shuttles that may drive along the track and carry material to and from the cells. We created a workflow, starting with the design of a production system model based on the IEC 62264 standard with an initial state description and the formulation of an envisioned goal state. Additionally, we generated a set of PDDL artifacts, one domain file and one to many problem files, corresponding to the number of provided goal state models. These files were handed over to an off-the-shelf PDDL solver that tried to find a sequence of actions leading from the initial state to the goal state (i.e., a so-called production plan). For this case study, we assembled toy trucks from a few sub-components. We could successfully proof (i) that the production system that has been designed is capable of re-sorting the transportation shuttles from a source sequence to a target sequence and (ii) how exactly the re-sorting can be accomplished, in terms of step-by-step directions.

1.5.3 Contribution 2: Temporal Model Management

For the contribution of Temporal Model Management, we elaborate the research directions listed below. These research directions have been identified by a systematic mapping study presented in Paper 6: Thirteen years of SysML: A systematic mapping study [146]:

Model Life Cycle Support: The results show that there is only limited support when using a modeling language such as SysML in the implementation phase, and very limited support for describing the whole life cycle of a system’s model from design to operation and backwards. These limitations are also discussed in Paper 7 [93], Paper 8 [150], and Paper 9 [96]. The SMS concludes that there is a need to exploit and adapt modeling languages (e.g., UML, SysML) for supporting the execution and analysis of systems during runtime as well as to align operational data with design model elements.

Modeling Hybrid Systems: Most of the selected publications in the SMS consider either discrete or continuous challenges when designing systems [8] [70]. This means that very rarely hybrid solutions in systems design are provided [56]. Therefore, further inves-
tigations should be undertaken for designing formal semantics for SysML to close the gap when combining discrete and continuous modeling and simulation. A challenge, we discuss in Paper 11 [94], Paper 12 [148], and Paper 13 [149].

Operational Semantics: Currently, there is no support, e.g., to shift property specification and verification tasks up to the model level. There is still a rule-based operational semantics missing to ensure a step-wise, state-based semantics, e.g., to describe a finite execution trace through a sequence of changes. In this context we present solutions in Paper 3 [110], Paper 8 [150], Paper 10 [95], and Paper 11 [94].

Paper 7: Towards Liquid Models: An Evolutionary Modeling Approach [93]

This paper presents initial concepts towards the extension of MDE by temporal aspects. Many of the ideas contained in this paper reflect our experience gained in the course of the InteGra 4.0 project (cf. Section 1.5.1). In this exploratory study, we conducted in-depth face-to-face interviews with managers as well as engineers of three software companies and nine companies from the areas of steel processing, wood processing, and paper production. These nine companies were a mix of small, medium-sized, and large companies, with varying economical dependencies. The diversity of the companies helped us to get a better understanding of open gaps between a system’s development and operation and motivated us to this paper.

Type of Contribution: We present early conceptual results of a unifying architecture for hosting so-called “liquid models”. This framework links design models to runtime concerns derived from distributed and heterogeneous systems during operation. We elaborate on proposed technologies for the layers of this architecture and identify research challenges ahead. At the same time the artifact represents a corner stone of the research work of Module 3 in the CDL-MINT project (see Section 1.5.1).

Validation of Contribution: We presented a first draft of an architecture, for stimulating a shift from isolated, one-shot, monolithic system descriptions to evolutionary, reusable artifacts. Based on this, we defined open research issues based on the identified open challenges derived from a comprehensive research of the state-of-the-art of various research fields (e.g., MDE, Process Mining, Software and Systems Engineering, etc.). Additionally, we presented a very first proof-of-concept of methods and techniques to address the challenges.

Paper 8: Model-Driven Time-Series Analytics [150]

In this paper, we move towards a well-defined mix of approaches to better manage the full life cycle of a system by combining prescriptive and descriptive model types. In particular, we introduce a model-driven time-series data analytics architecture for
1.5 Contributions to Temporal Model-Driven Systems Engineering

harmonizing model-driven and data-driven approaches. Based on this architecture, we show how data analytics works for modeling languages using standard metamodeling techniques. This means, design-oriented languages are extended for representing runtime states as well as runtime histories, which in turn allow the formulation and computation of runtime properties by employing the Object Constraint Language (OCL) \cite{108}. The extensions needed on the metamodel level are non-intrusive and related to existing approaches for specifying the operational semantics of languages. The presented runtime history metamodel fragments are applicable for any modeling language used during design (e.g., UML, SysML) comprising features to be measured and events to be tracked as the current metamodeling languages Ecore and OCL are reused.

Type of Contribution: We present a unifying architecture, as concrete metamodel, for a design language to cope with a model-driven as well as data-driven perspective on systems. This architecture builds on the classical MDE approach by modeling a system downstream in terms of code generators, but at the same time supports an upstream in terms of mapping data back to design models. At the metamodel level (cf. Figure \ref{fig:architecture} at the Language Layer), the design language is defined with the help of a metamodeling language, which is in our setting Ecore. Conforming to the design language, the design models are defined at the model level describing the static (i.e., structural) as well as dynamic (i.e., behavioral) aspects of a system to be developed. For the vertical transition from the model to the realization level, we assume the existence of an M2T-transformation code generator, as presented further in Paper 10, Paper 11 as well as Paper 12. In a next step, we generate a so-called runtime observer from the design model. The runtime observer collects important information from the running system to represent the current state of the system. Those observations should not only be recorded by observing the running system, but should also be representable at the model level. Thus, we extend the design language with a dedicated runtime language. This metamodel defines the syntax to represent snapshots of the running system connected to the design model elements. Those snapshots are represented in the so-called runtime state models which extend the design models and may be directly updated by the observer during runtime.

In summary, our architecture is used to monitor a system on the model level. In a first step, we map runtime data at the model level for one single point in time. In a next step, we define the runtime history of a system. For reasoning purposes, it is important to have the complete history of value changes as starting point, since a single snapshot is definitely not sufficient for giving useful insights of the operation of a system. Thus, in the time-series database, we store the observations of the running system. Based on those collected observations, the runtime history models may be directly updated. These models conform to the runtime history language, which is an extension of the runtime language. In the runtime history language, the syntax is defined for representing histories of runtime phenomena of interest, e.g., property values, events, etc. Finally, after
defining those concepts for storing observation histories at the model level, it is also possible to analyze those observations. For this purpose we define runtime properties based on OCL by introducing derived properties for the metamodel elements.

**Validation of Contribution**: Our recurring example of a lab-sized five axis grip-arm robot is used for evaluating the pertaining temporal model management and data-driven model analytics. We validated the concepts of the proposed design modeling language, i.e., extensions for runtime states, runtime histories, and runtime properties, as reusable metamodeling “blueprints”. We demonstrated the time series analysis regarding property value changes of the gripper’s axis angles during operation in a laboratory-like environment of a sorting system. The implementation and evaluation results can be found on our project page [97].

**Paper 9: Temporal Models on Time Series Databases [96]**

Building on the temporal architecture presented in Paper 8, we extend the approach for enabling partial mappings from metamodels and their instances (i.e., models) to time-series databases. In addition to time-series representations, this special type of temporal databases enables time-series analytics in order to deal with additional activities in engineering technical systems, an emerging trend we surveyed in Paper 6. In the presented approach, we allow for model simulation runs which may be analysed by time series analytics as well as for model-based runtime monitoring of systems reporting their changes and states to time series databases. We demonstrate both scenarios in a production system case study and evaluate in particular two mapping strategies with respect to the required data storage and query answering performance.

**Type of Contribution**: For combining models, especially EMF-based models, with a Time Series Database (TSDB), we aim for a polyglot solution where the static information resides in the model as it is already available, e.g., by XMI or other model persistence mechanisms, and only the time-sensitive information is stored in the TSDB. The two storage parts are combined by a ModelAPI that can be accessed and used by various applications. This unifying API abstracts implementation details and allows for a similar way of working with models as it is provided by EMF out-of-the-box. In particular, we reuse as much as possible and only extend those parts which are really required. As a result, the model sticks as close as possible to the EMF standard and the required information for the TSDB is attached in a light-weight manner. As design choice for implementing the unifying ModelAPI with a polyglot, we propose in a first step a so-called Time-Series (TS) profile, realized with EMF annotations, for extending existing metamodels by time-series aspects. This TS-profile defines different kinds of stereotypes for annotating classes, structural features as well as operations. Based on the usage of the profile, we present the so-called Model-to-Time Series Mapper (M2TS-mapper) and two mapping strategies for this mapper: (i) to store each temporal
property individually (strategy of single property mappings), and (ii) to store the whole object with all its associated information (strategy of complete object mappings). For instance, the first one enables to map a single property like the temperature of a room in order to continuously log the progression of the property value in the TSDB and to query it. By employing the second one, individual properties do not have to be annotated as temporal features (like in case of single property mappings), but the containing classes, and thus, the associated objects with their properties are stored in the TSDB as measurements. On the basis of the TS-profile and the applied mapping strategy, we provide appropriate query capabilities in a further step. These query capabilities enable the M2TS-mapper not only to inject data to the TSDB, but to extract data from the TSDB by model-based queries. As the derived runtime properties are in essence standard-derived properties, they can be simply reused in standard OCL queries.

Validation of Contribution: For evaluation purposes we validated (i) the scalability of database sizes and (ii) the performance of runtime queries. For (i), the TSDB size on the basis of model changes showed that both strategies had a linear increase. We recognized that the size of the database for the complete object mapping strategy increased slightly faster than for the single property mapping strategy. This could be explained by the fact that whenever a value of a property changed, the entire object was stored with a new timestamp. Regarding the performance of runtime queries ((ii)), the queries were fast (from 1ms to about 7ms) depending on the entries in the TSDB. However, as the size of the database increased, the queries for MeanMaxMode and GetValueAt for the single property mapping became slightly slower than in the case of the complete object mapping. This could be explained by the fact that starting from a certain number of entries, it plays a role whether the possible results have to be selected first, or are already selected and only need to be screened. Finally, the hypothesis testing (i.e., Wilcoxon rank-sum testing [130]) showed that there was no significance regarding the difference between the two mapping strategies.

1.5.4 Contribution 3: Data-Driven Model Analytics

During the early phases, MDE approaches are frequently used to design systems, whereas during operation data-driven approaches are used to reason about the system’s behavior based on data logs. The main challenge is to establish so-called trace links between the initial design model and the operation model for (i) monitoring meaningful data from operation, (ii) aligning data logs with the design model for providing a semantic anchoring of data, as well as (iii) providing meaningful analytics to reason about improvements or the fulfillment of given requirements. Additionally, it must be considered that the data acquisition is taking place during operation time. As a result, the data is not a finite set, but a continuous stream of data.
We present an initial unifying framework to combine MDE with Process Mining (PM) techniques on the basis of previous research work presented in [92]. Thereby, not only the dependencies between different process steps may be uncovered, but also, dependencies between data and process steps are approachable. In this Paper 10, we are focusing on prescriptive models needed for realizing a system and descriptive models used for describing a system as it is actually realized in a runtime environment. For aligning these two kinds of models, we introduce an approach which we call execution-based model profiling. Thereby, model profiles are automatically generated from execution logs of running systems. In particular, we define execution-based model profiling as a continuous process to generate observation models during a system’s operation and to check whether these models correspond to the initial design model or not. The approach is based on executable modeling languages which provide operational semantics for interpreters. It further provides translational semantics in form of code generators to produce code for a concrete platform to realize a system (see Figure 1.1 Language Layer to Data Layer). This contribution follows one of the research directions identified by our survey in Paper 6.

Type of Contribution: In order to combine the prescriptive perspective with the descriptive one, we introduce a first conceptual unifying framework and an operational language that acts as a logging metamodel at the language layer (cf. Figure 1.1). This observation language determines which runtime changes should be logged (e.g., state changes, attribute value changes) by the proposed stereotype «observe». This stereotype has to be annotated in the design model. Thus, the metamodel defines the syntax and semantics of the data logs we observe from a running system. These execution logs are stored as observation models which conform to our observation language. These models could be then considered at the mining layer (see Figure 1.5.1) as input for any kind of analytical tool, for instance, to check non-functional properties (performance, correctness, appropriateness).

Validation of Contribution: We evaluated the unifying framework by an explanatory case study based on a traffic light system example where we combined MDE techniques with Process Mining (PM) techniques [127]. For the purpose of validating the transformability of the approach, the outcomes show that the operational semantics of the modeling language is rich enough to automatically derive observation metamodels from log files. The results of validating the interoperability of the observed models show that they fulfill the requirements of general workflow-oriented formats of PM tools. In this paper, we employed the open source tool ProM Lite in Version 1.1 [126]. For runtime verification (usefulness), we applied the $\alpha + +$ algorithm of ProM Lite to derive a Petri net. The generated net corresponded to the initial state machine. Thus, we could demonstrate that the state machine was realized by the code generator as intended at
1.5 Contributions to Temporal Model-Driven Systems Engineering

design time. For the detection of timing inconsistencies (timeliness), we filtered the sequence of transitions by an ATL transformation and analyzed it with the performance plug-in of ProM Lite. The inconsistencies between the specification and implementation levels were within a range of milliseconds (40 to 190 milliseconds average delay per transition). In a final step, we demonstrated that the average values of the delays could be propagated back to the design model so that the timing becomes more precise during system execution.

Paper 11: Reverse Engineering of Production Processes based on Markov Chains

New technologies, such as IoT, enable us to continuously observe running systems based on sensor data, which helps us to get a clear picture of the current status of a system based on the gathered data during the execution of processes. Since, such raw data are streams of performed actions, we need mechanisms to transform those streams into so-called descriptive models for further analysis. Therefore, we present an automated reverse engineering approach based on Markov chains that combines model-based downstream information derived from prescriptive models with sensor-based upstream information of a lab-sized production line during operation.

**Type of Contribution:** We present a reverse engineering approach by computing behavioral models from timely observations based on the system components’ emission in form of operational logs. These logs reflect the activities happening in a system during operation. In particular, we consider the duration time of operations applied on work items (e.g., resource-specific operation calls). We transform these logs to Markov chains. This enables us to deal with the complexity of runtime information as well as to provide reasoning mechanisms for future adaptations. The approach bases on previous research work in the field of Web engineering [6], where we additionally identified trends for user analysis in this field.

**Validation of Contribution:** The procedure, languages, and tools have been evaluated based on a lab-sized production system, which is hosted at IAF of the Otto-v.-Guericke University Magdeburg [68]. One main focus was on feasibility and usefulness of the introduced modular transformation chain. Furthermore, we analyzed workload characteristics and bottlenecks when using the IAF plant in certain settings by statistical performance measurings. All artifacts of this evaluation can be found on our project page [3].

Paper 12: Automatic Reverse Engineering of Interaction Models from System Logs

During the execution of software when the system is operating, the executed operations can be traced based on sensor value streams or logging code. This enables to derive the
behavior of the system based on the communication of system components. Usually, such log traces have the form of huge text-based files which are difficult to work with. As a consequence it is not straightforward to fully track and understand the interaction and communication between the components. In this paper, we tackle this challenge by presenting a scalable reverse engineering approach to automatically transform log traces to an appropriate and user-friendly graphical representation. For this purpose we use established methods and techniques from MDE as basis (cf. Section 1.2) to provide an end-to-end traceability from design to runtime and backwards.

**Type of Contribution:** The artifacts are a so-called log-metamodel for enabling an object-oriented representation of system logs and an architecture to automatically reverse engineer interaction models in terms of UML sequence diagrams. This architecture is divided into three parts: (i) the creation of object-oriented interaction models in form of sequence diagrams derived from executed operations, (ii) the alignment between those sequence diagrams and the corresponding prescriptive models (created at design time) by so-called trace links, and (iii) the creation of a runtime profile and its display in the corresponding design models. The log-metamodel and the architecture enable an object-oriented view on executed operations and help to trace the inter-object communication among system components. Additionally, the profiled information could be back-propagated to the initial design model for reactivity purposes.

**Validation of Contribution:** The procedure and prototypical implementation were evaluated on the basis of a case study by employing a self-driving car. We based this case study on the guidelines of Runeson and Höst [113]. The results show that we are able to generate interaction models from system logs based on a T2M-transformation and that the relationships between runtime models and design models could be established by means of generated queries. However, the scaling of modeling tools is still a crucial issue. The strength of our approach is that we can keep the relevant information in a unified modeling language, namely UML. Thus, design tools can be reused with their integrated tooling and there is no need to learn new technologies or languages for analyzing runtime information, which is a benefit for domain experts. For presenting the case study as well as the results, we have provided a project page [145].

**Paper 13: Model-driven Runtime State Identification [149]**

In this contribution, we combine MDE techniques with Time-Series Database (TSDB) (see Section 1.5.3) and Process Mining (PM). Thereby, we take up the challenge to continuously listening to value streams in order to determine whether a state has indeed occurred, i.e., if the specific combinations of variable values have occurred over all streams at the same time. In particular, the realization precision of systems as well as measuring inaccuracies complicate this process as false positives and false negatives may occur when matching state templates to data streams. Based on first ideas pre-
presented in [147], we address this challenge by introducing a novel approach where we automatically generate state realization event queries derived from state machines for an appropriate state identification at runtime. This enables us to continuously observe multiple data streams of distributed sensor devices in order to identify a system’s entire state during runtime. The approach enables to automatically transform behavioral models (i.e., state machines) into time-series queries for matching sensor value streams with pre-defined variable values of the design model in order to report identified states from execution. Additionally, the approach provides a recording mechanism, an abstraction part, and a runtime analysis.

**Type of Contribution:** We present a metamodel named **MD-RISE** which we prototypically implemented. For this purpose we have a number of prerequisites that must be met: (i) the system’s workflow must be expressible by means of a state machine, (ii) the different states of the system must be unambiguous that values describing a state are not identical for two different states, (iii) numeric values must be returned by sensors at runtime and must be storable in a TSDB, and (iv) the time stamps must be accessible. Based on this prerequisites we motivate the approach by an example of a five axes grip-arm robot (gripper). The gripper is an automation system consisting of a controller, sensors, and actuators. At design time, we model the structure and behavior of the gripper by using a subset of SysML, namely the block definition diagram and a state (machine) diagram [125]. Additionally, we consider for each property a specified tolerance range (based on expert knowledge) that defines an acceptable deviation of the assigned property values during runtime. Such deviations may occur due to sensor delays, measurement inaccuracies, etc.

Based on the metamodel, we automatically derive a query on the basis of the state machine, a so-called **state realization event query**. This query helps to identify states based on the recorded sensor value streams in the TSDB. For this we use an M2T-transformation (cf. Section [1.2.2]) to automatically transform model elements to query statements in form of text strings. The thereby identified states contain information as follows: the actual time in the granularity of seconds (i.e., timestamp) and the recognized state. In a next step, we generate a state-based log model that consists of the information of all identified states, and additionally, a case ID for identifying the corresponding process instance. Such a case ID is required when using PM tools in order to distinguish different executions of the same process. We employ this case ID in our approach to identify single runs of the state machine when executed. In a further step, the state-based log model is transformed to an event-based log model by employing an M2M-transformation. Similar to the approach presented in Paper 10, we use ProM Lite in Version 1.1.

**Validation of Contribution:** The transformation, event queries and tool support were evaluated by means of a case study of a laboratory-sized five-axis gripper arm robot. This setting allowed us to analyze the (i) correctness of the identified states, (ii) com-
pleteness of the identified states, and (iii) performance of the queries. The outcomes show a linear increase as well as good precision and recognition values, but depending on the tolerance range as well as distinctness of the states. The case study design followed again the guidelines defined by Runeson and Höst \cite{113}. The study and its results are published on a project page \cite{144}.

Paper 14: From AutomationML to AutomationQL: A By-Example Query Language for CPPS Engineering Models \cite{141}

In this paper, we extend the framework of Paper 10 by a query component. By following the main principles of the Query By-Example approach (QBE), we present a novel AutomationML query approach for querying AutomationML models at the mining layer (see Figure 1.1 on top) by formulating queries by-example as model fragments. In particular, we propose a dedicated query language for AutomationML called AutomationQL (AQL), which is directly derived from AutomationML. Using this query language, queries can be defined in a QBE manner which allows engineers to formulate queries in terms of AutomationML concepts instead of being burdened when switching to an implementation-oriented query language. Thus, the engineers are as close as possible to modeling languages they usually work with.

**Type of Contribution:** We present AQL which is a graph pattern-based query language based on concepts of AutomationML. AQL supports positive and negative graph patterns as well as the computing of transitive closures to investigate recursive tree structures and to match for element sets. The query results are explicitly represented in a result model which acts as a proxy to the AutomationML base model elements.

**Validation of Contribution:** We implemented a prototype of AQL in Eclipse, based on the CAEX workbench, which provides tool support for AutomationML in Eclipse \cite{87}. In particular, we specified AQDL and AQRL as Ecore-based metamodels. Using the standard EMF capabilities, we generated tree-based modeling editors for both languages. For executing AQDL queries on AutomationML models, we implemented a prototypical interpreter in Java. The interpreter read the AutomationML models and AQDL models and produced AQRL models as output. For demonstration purposes we employed the Pick and Place Unit (PPU) demonstrator \cite{107} hosted at the Institute of Automation and Information Systems at TUM. We instantiated each language feature by defining a specific query (Q1 - Q8) requiring this feature. We figured out that AQL supports positive and negative graph patterns, computing transitive closures to investigate recursive tree structures and to match for element sets. The query results were explicitly represented in a result model. We have provided an open source implementation of our prototype with further description and examples on our project website \cite{142}.
1.6 Summary

This cumulative habilitation thesis follows a temporal model- and data-driven approach in the application field of Systems Engineering. Thereby, automation capabilities are provided for reusing design models from the very beginning, which are then during a system’s operation or simulation continuously augmented with runtime data. In addition, we provide dedicated services for a continuous planning, model mining, and issue management. Thereby, we reduce time consuming manual tasks for (i) figuring out appropriate model elements to reuse, (ii) checking consistency, (iii) searching for positive and negative modeling patterns, and/or (iv) providing specific views for particular stakeholders. For this purpose, the explicit consideration of temporal aspects not only on the model, but also on the level of model elements is essential. We provide a models “in-the-loop” approach from design to operation and backwards by combining downstream information from the MDE-process with upstream information gathered at runtime. This enables a shift from isolated one-shot system prescriptions to evolutionary models populated by timing aspects, where the focus is not only to represent the current state to steer the system, but on the representation of the system’s history.

The presented contributions are all following a design science methodology considering the design, implementation, and evaluation of artifacts (i.e., frameworks, models, and methods) in the context of two application fields, the stationary industry (i.e., manufacturing) and the AEC-industry. In the first of the three contribution clusters, we focus on the handling of heterogeneous sources distributed at various layers in the automation pyramid and beyond (cf. Section 1.4.1). For overcoming connectivity and integration shortcomings (cf. Figure 1.1, bottom down), the presented papers (cf. Papers 1 to 5) in this section deal with the combination and revamping of data from heterogeneous sources into new and harmonized representations to provide a nearly seamless vertical (from data sources downwards and upwards along the layers) and horizontal (from one system to another at the same layer) exchange of relevant data. The second cluster, Temporal Model Management (cf. Section 1.5.3), considers temporal aspects of models. In Paper 6, we identified this temporal aspect as a missing link. Accordingly, we developed and implemented a unified temporal model framework as presented in the Papers 7 to 9. These contributions address challenges regarding hosting runtime monitoring and handling temporal models (cf. Section 1.4.2). For this purpose, we provide query facilities for reasoning about various model element aspects over time (e.g., Paper 9). In the third cluster, we provide Data-Driven Model Analytics (cf. Section 1.5.4) to overcome the challenge presented in Section 1.4.3. We establish so-called trace links between the initial design model and the operation model for (i) monitoring meaningful data from operation, (ii) aligning data logs with the design model for providing a semantic anchoring of data, as well as (iii) providing meaningful analytics to reason about improvements or the fulfillment of given requirements (cf. from Papers 10 to 14).
1 Introduction

All of these contributions were evaluated by observational, analytical, experimental, and descriptive methods by following the guidelines of Runeson and Höst [114]. As main evaluation environment, we built up a lab-sized automation system environment in form of a five axes grip-arm robot, around which we set up various demonstration cases. In addition, we employ a traffic light system and a self-driving car as demonstrators, which we have developed with support of a project partner of the CDL-MINT laboratory. Furthermore, we use lab-sized test-beds from our research partners such as the lab-sized production system IAF of the Otto-v.-Guericke University Magdeburg (as presented in Paper 11) and the CTU’s Czech Institute of Informatics (as presented in Paper 5).

Table 1.1: Meta-information summary of papers included in this thesis

<table>
<thead>
<tr>
<th>Paper</th>
<th>FFG InteGra 4.0</th>
<th>FFG DigiTrans 4.0</th>
<th>CDG CDL-MINT</th>
<th>BMBWF TransIT</th>
<th>Model-Driven Connectivity (MDC) / Temporal Model Management (TMM) / Data-Driven Model Analytics (DDMA)</th>
<th>Collaborations</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>X</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>MDC:: Procedure &amp; Notation</td>
<td>Industry: Eisenstraße Niederösterreich, Zukunftsaakademie Mostviertel</td>
</tr>
<tr>
<td>2</td>
<td>-</td>
<td>X</td>
<td>X</td>
<td>-</td>
<td>MDC:: Technique &amp; Descriptive Model</td>
<td>-</td>
</tr>
<tr>
<td>3</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>X</td>
<td>MDC:: Analytic Model &amp; Notation</td>
<td>Scientific: MUL, TU Wien Industry: buildingSMART International</td>
</tr>
<tr>
<td>4</td>
<td>-</td>
<td>X</td>
<td>-</td>
<td>-</td>
<td>MDC:: Procedure &amp; Prototype</td>
<td>Scientific: CTU Prague, TU Wien</td>
</tr>
<tr>
<td>5</td>
<td>-</td>
<td>-</td>
<td>X</td>
<td>-</td>
<td>MDC:: Technique &amp; Tool</td>
<td>Scientific: Software Competence Center Hagenberg (SCCH)</td>
</tr>
<tr>
<td>6</td>
<td>-</td>
<td>-</td>
<td>X</td>
<td>-</td>
<td>TMM:: Descriptive Model</td>
<td>-</td>
</tr>
<tr>
<td>7</td>
<td>X</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>TMM:: Procedure</td>
<td>-</td>
</tr>
<tr>
<td>8</td>
<td>-</td>
<td>-</td>
<td>X</td>
<td>-</td>
<td>TMM:: Technique &amp; Notation &amp; Analytic Model</td>
<td>-</td>
</tr>
<tr>
<td>9</td>
<td>-</td>
<td>-</td>
<td>X</td>
<td>-</td>
<td>TMM:: Technique &amp; Notation &amp; Descriptive Model</td>
<td>Scientific: Universitat Oberta de Catalunya</td>
</tr>
<tr>
<td>10</td>
<td>-</td>
<td>X</td>
<td>-</td>
<td>-</td>
<td>DDMA:: Notation &amp; Descriptive Model</td>
<td>Industry: Lieber Lieber Software GmbH</td>
</tr>
<tr>
<td>11</td>
<td>-</td>
<td>-</td>
<td>X</td>
<td>-</td>
<td>DDMA:: Analytic Model</td>
<td>Scientific: University Magdeburg</td>
</tr>
<tr>
<td>12</td>
<td>-</td>
<td>-</td>
<td>X</td>
<td>-</td>
<td>DDMA:: Analytic Model &amp; Prototype</td>
<td>Industry: Lieber Lieber Software GmbH</td>
</tr>
<tr>
<td>13</td>
<td>-</td>
<td>-</td>
<td>X</td>
<td>-</td>
<td>DDMA:: Technique &amp; Analytic Model &amp; Prototype</td>
<td>Scientific: Practical Robotics Institute Austria (PRIA)</td>
</tr>
<tr>
<td>14</td>
<td>-</td>
<td>-</td>
<td>X</td>
<td>-</td>
<td>DDMA:: Technique &amp; Notation</td>
<td>-</td>
</tr>
</tbody>
</table>
To give a condensed overview, Table 1.1 summarizes the presented papers accumulated for this thesis and their assignments to research projects, general research fields based on the classification of result types as presented in [122], and scientific as well as industrial collaborations.

1.7 Outlook

While having the above mentioned contributions as an important cornerstone for Temporal Model-Driven Systems Engineering, there are several other concerns that are not discussed in this habilitation thesis, but are currently investigated by the applicant as presented by a research roadmap.

First of all from a runtime perspective, several challenges have to be tackled to realize an efficient monitoring process of system components, since (i) not all relevant parameters are directly observable, (ii) parameter values keep changing during observation—a fact that is known as concept drift, and (iii) the observation has to be performed while the system operates, to name just a few.

Second, an interoperable tool chain is needed starting from engineering, over simulation, to operation and in the reverse direction by back-propagating runtime data and experiences from operations to engineering. Based on the investigated temporal model framework, groundbreaking foundations as well as applied research are required to further detail and realize them in the direction of Digital Twin Platforms and Model-Driven Digital Twin Engineering. We present a preliminary reference architecture in this direction in [84, 83]. However, using digital twins also poses new requirements on traditional software engineering practices [112].

Although existing digital twin platforms of big cloud providers (e.g., Azure [101], Eclipse [44], or AWS [4]) provide a lot of benefits to practitioners, the creation and maintenance of digital twins still involves a lot of (human) effort. Information about the system must be entered into different tools that comprise the digital twin, and synchronized with the system to send data to the correct digital twin. Even if this is done, every change in the system requires changes at different positions in its digital twin to ensure consistency. Different research works [77, 118] show that MDE techniques can already be used to automate the construction of a digital twin. However, there is still little knowledge about digital twins of evolving systems.

Third, a broader application context has to be considered already starting in the early phase of requirements elicitation by additionally taking non-functional properties into account. This would be useful, e.g., to better estimate the sensitivity of variables based on external factors such as different workloads, product types, etc. We have already started research work in this direction, as presented in [91, 95] or [94].

Fourth, in the AEC industry, a wide variety of different actors with vastly different background and perspectives are involved during a project. This has also lead to a
variety of isolated software solutions as well as a slew of different artifacts and types of information with little to no collaborative access. In addition to different tools, internal company standards, best practices, and guidelines complicate collaboration on infrastructure projects. Accordingly, there is a need for a common temporal model framework for hosting, versioning, and querying those various artifacts. While existing versioning management systems such as Git and CSV provide support for storing, retrieving, and keeping track of different versions of source code and documents, they lack support for organizing different types of artifacts, managing dependencies between artifacts, and providing adequate role and permission models for handling complex workflows in an environment with highly diverse stakeholders. Therefore, a collaborative platform for industry domain experts (with little to no computer science background) is needed to store and easily maintain different variants and versions of a multitude of diverse artifacts (models, paper made notes, 2D construction plans, spreadsheets, 3D models, etc.). Furthermore, providing sophisticated query and retrieval mechanisms across the different artifacts (and their respective versions and variants) is key to the digital transformation in the AEC domain.

1.8 Bibliography


2 From business functions to control functions: Transforming REA to ISA-95

A. Mazak and C. Huemer;
Proceedings of the 17th IEEE International Conference on Business Informatics (CBI),
DOI: 10.1109/CBI.2015.50
From business functions to control functions: Transforming REA to ISA-95

Alexandra Mazak
TU Vienna
Austria, Vienna
Email: mazak@big.tuwien.ac.at

Christian Huemer
TU Vienna
Austria, Vienna
Email: huemer@big.tuwien.ac.at

Abstract—In the context of smart factories, a seamless information exchange between information systems on the same layer (horizontal integration) and between information systems on different layers (vertical integration) is a key issue. For this purpose we aim for an integrated modeling framework spanning over production chains and value networks. In building this framework, we first concentrate on the layers realizing the business functions and the manufacturing control functions. Thereby, we build up on the Resource Event Agent (REA)-business ontology (ISO/IEC 15944-4) to describe external activities requiring horizontal integration with business partners and internal activities serving as a hook for vertical integration within a manufacturing enterprise. Furthermore, we base our framework on the ISA-95 industry standard (ANSI/ISA-95; IEC 62264) to describe the vertical integration within an enterprise. In this paper, we demonstrate how information given in REA models is transformed to corresponding ISA-95 skeletons. In other words, we show how a model describing the main business functions of an enterprise is used to derive essential concepts relevant to the manufacturing execution system.

I. INTRODUCTION

The German working committee for Industrie 4.01 has identified among others the following research issues [1]:

- horizontal integration through value networks
- vertical integration of networked manufacturing systems
- end-to-end digital integration of engineering across the entire value chain

Industrie 4.0 use case scenarios relating, e.g., to networked manufacturing, self-organizing adaptive logistics, and customer-integrated engineering will require business models that will primarily be implemented by what could be a highly dynamic network of businesses rather than by a single company (e.g., to link products of a manufacturing company with appropriate services provided by another company) [1]. On the one hand—for realizing a horizontal integration through value networks—we need appropriate language constructs to describe business relationships between companies also taking their different business views into account. On the other hand—to enable a seamless vertical integration of networked manufacturing systems—we need a fundamental understanding of activities and information flows within manufacturing companies. However, information flows between the horizontal layer (business partner networks) and the vertical layer (from an ERP system to a Manufacturing Execution System) are very limited or not even possible at all [1]. IT systems still tend not to cross company or factory boundaries. The German initiative for Industrie 4.0 points out that the use of information technology in this context has largely failed to reflect the existence of manufacturing networks. One problem is, among others, that value chains (from customer requirements to production and distribution) tend to be relatively static since they often have been created over many years.

From a technical as well as an economic perspective an end-to-end digital integration will be a key issue to realize smart factories. This integration will enable all parts of a manufacturing company (enterprise level), shop floor control level, and shop floor level) to be connected to each other through a global information system with customers, suppliers, and other external participating parties. The potential of an end-to-end integration is huge. For example, this will allow in future to individual, customer-specific criteria to be included in the design, configuration, ordering, planning, manufacture and operation phase. This will enable last-minute changes to be incorporated and very low production volumes (batch size of 1). The realization of this ambitious goal requires appropriate interfaces for integrating the individual subsystems [2]. However, it is still common practice that IT systems exchange information through extensive interfaces, but can only utilize specific pieces of that information. The situation is further worsened by the problem that many different interfaces introduce dependencies whose management can become complex and hard to achieve. Thus, the system complexity will rise drastically.

There is still a lack of appropriate concepts for interface integration by which different operational layers can be connected for communication. However, to provide a universal infrastructure for a seamless information exchange is crucial for a successful implementation of the Industrie 4.0 initiative. Modeling can act as an enabler for managing this integration. Models are representations of real and hypothetical scenarios that only include those aspects that are relevant to the issue under consideration. The working group of the German initiative points to the fact that “the use of models constitutes an important strategy in the digital world and is of central importance in the context of Industrie 4.0” [1]. For this purpose appropriate language constructs are required to formally describe the increasing functionality, increasing

1Please note, that the approach introduced in this paper is aligned with the German initiative “Industrie 4.0”, and therefore, we do not translate it to the English term “Industry”.
product customization, dynamic delivery requirements, and the rapidly changing forms of cooperation between different companies in order to provide end-to-end transparency.

II. APPROACH

The approach presented in this paper is based in its orientation on the recommendations of the German working committee for Industrie 4.0 which was released in 2013. Amongst other things, the working committee points out that production systems are to be linked vertically with business processes within decentralized production sites and enterprises, and that they are to be distributed horizontally among suppliers, distributors and customers. In order to meet these requirements, we aim for an integrated modeling framework spanning over the horizontal layer (value networks) and the vertical layer (production chains). For this purpose we do not intend to start from scratch by defining our own all-encompassing modeling language. In contrary, we want to build on existing well-accepted modeling languages.

The German working group defines the vertical integration as “the integration of the various IT systems at the different hierarchical levels (e.g., the actuator and sensor, control, production management, manufacturing and execution and corporate planning levels in order to deliver end-to-end solution”), [1]. We consider the concepts and models of the industry standard ISA-95 (ANSI/ISA-95; IEC 62264) [3], [4] as appropriate to model the vertical integration of information flows between the different levels within an enterprise. ISA-95 is an international standard released by the International Society of Automation for developing an automated interface between Enterprise Resource Planning Systems (ERP) on the enterprise level and Manufacturing Execution Systems (MES) on the shop floor (control) level. Based upon this standard, which consists of five parts, the standard IEC 6266 was established.

In analogy, we apply concepts of the Resource-Event-Agent business ontology (REA) (ISO 15944-4) [5] which allows describing the interfaces between the systems of different business partners as a horizontal integration of information flows. The German working group defines the horizontal integration as referring to “the integration of the various IT systems used in the different stages of the manufacturing and business planning processes that involve an exchange of materials, energy and information both within a company (e.g. inbound logistics, production, outbound logistics, marketing) and between several different companies (value networks)”, [1]. In a business environment, REA is used to identify the value adding activities of the company. In general, value adding activities are either transformations of resources by producing something or transfers of resources by exchanging something with an external party. In other words, REA is able to provide the binding clue between the internal production processes requiring vertical integration and the external trading activities requiring horizontal integration.

In our approach, we elaborate on a seamless integration of the horizontal and vertical layers which implies that necessary information must flow between these layers. For realizing a vertical as well as a horizontal integration through value networks appropriate language constructs are needed to describe interface integration within the company between different kinds of IT systems (ERP, MES) at different levels and between multiple enterprises and various participating parties (vendors, sub-contractors, customers). For this purpose we transform REA concepts to ISA-95 concepts. Our approach is independent of any software solution. In fact, companies may use different ERP and MES systems and still have to collaborate with each other.

Following our aim of an integrated modeling framework by transforming concepts of REA to concepts and models of ISA-95, we concentrate on these two standards in Section III on related work. Section IV presents the REA meta model and its core concepts. In Section V, we present the ISA-95 meta model. Section VI provides the core of our paper describing the transformation rules from REA to ISA-95. This transformation is illustrated by examples in Section VII and Section VII-B. We close the paper with a summary of our contribution in Section VIII.

III. RELATED WORK

A. Industry Standard ISA-95

The ISA-95 standard has been developed for global manufacturers, i.e., a production company with decentralized, networked production plants. This standard fosters a universal communication within a manufacturing company (headquarters and distributed industrial premises). ISA-95 can be applied in all industries, and in all sorts of production processes like batch processes, continuous processes, and repetitive processes. ISA-95 was specifically developed for creating interfaces between the enterprise domain with its ERP system at Level 4 and the shop floor control domain with its MES at Level 3 and lower (Levels 2, 1, 0). It offers a fundamental understanding of activities and information flows within a manufacturing company. The standard describes hierarchy models which are based on the Purdue Enterprise Reference Architecture (PERA) for Computer Integrated Manufacturing (CIM) [6].

Figure 1 shows in a simplified manner the different levels of the functional hierarchy model. In addition, the equipment (e.g., site, area, process cell, production line, storage zone) are usually organized in a hierarchical fashion. The red cycle in Figure 1 shows the enterprise-control interface between Level 4 and Level 3. Between these levels the standard points to 31 information flows, as outlined in Figure 2. The wide dotted line of this functional enterprise control model illustrates the boundary of the enterprise-control interface. Everything that lies outside the dotted lines belongs to Level 4, and everything that lies inside the dotted lines belongs to Level 3. The
labeled lines indicate the 31 information flows of importance to manufacturing control. The model contains 12 functions [3].

ISA-95 describes step by step the tasks of each of these functions. The functions shown in rectangles (e.g., research, development and engineering, marketing, sales) are external entities and as such they are not described in the functional enterprise control model. These entities are components outside the boundaries of this model that send data and receive data from the functions. The basic data to be exchanged in this model are information flows which are defined by ISA-95 for the sectors personnel, material, equipment, physical asset and process segment. The process segment is a logical group of equipment, physical asset, personnel, material required to carry out a specific part of a process (e.g., mixing, sawing, etc.). These sectors are defined as object models in ISA-95 which constitute basic building blocks with which the information flows of the functional hierarchy model are constructed (cf. Figure 1). In order to standardize the 31 information flows between Level 4 and Level 3 ISA-95 groups them into four categories: (i) production capability information, (ii) production definition information, (iii) production schedule information, and (iv) production performance information [3].

B. Resource-Event-Agent Business Ontology

The Resource-Event-Agent Business ontology (REA) was developed by William McCarthy [7] for the application-independent description of economic phenomena (i.e., exchanges which can either be transfers or transformations of resources). The acronym REA stands for the three main concepts of the ontology Resource, Event, and Agent. Agents are persons, companies, or organizational units capable of having control over resources, who/which participate in an economic exchange. Resources are transferred or transformed during an economic exchange. Resources can be goods, material, rights, labor, equipment, physical assets or services which agents have control of and which should be monitored and controlled in a business environment. An event is considered as a class of phenomena reflecting exchanges of resources. REA has its roots in the accounting discipline and is based on strong concepts of the literature in economic theory [8]. Additionally, REA focuses on IT implementation issues and follows a conceptual modeling approach [9]. This makes it a good choice for being used in a business model-driven engineering approach. Moreover, the REA business ontology is a wide accepted language in the academic world to design enterprise information systems. For instance, in the ISO/IEC 15944-4 Open-edi standard [5]—which addresses business communications between enterprises—REA is used as an ontological framework for specifying concepts and relationships involved in business transactions and transactions. REA initially focuses on concepts of economic exchanges of the present and the past.

IV. THE REA META MODEL

In this section, we elaborate on the REA meta model. Thereby, we build up on previous work [10], [11], [12]. In these papers we developed a domain specific language (DSL) for the REA ontology called REA-DSL. The REA-DSL provides a formal definition of the REA language concepts by means of Object Management Group’s (OMG) meta-modeling architecture called Meta-Object Facility (MOF) [13]. MOF comes with a meta-meta model (M3 layer) that allows us to define the REA concepts as a meta-model (M2 layer). In this section, we introduce the existing REA concepts by means of meta-models and also show some additional extensions required for this work.

REA consists of three different layers concerning entrepreneurial logic and details at a different level of granularity. The three layers from top down are:

1) value chain specification layer
2) duality specification layer
3) task specification layer

In the following subsections, we explain the meta-models of these REA layers.

A. REA Value Chain

A business model defines how a company creates value. It specifies a competitive strategy by looking at those activities that create value for the company. A seminal work in this respect has been Michael E. Porter’s book “Competitive Advantage” [14] in which he first introduces the concept of the value chain. A value chain is a set of activities that an organization carries out to create value. Porter proposes the concept of a value chain to examine all of a company’s activities, and see how these are connected.

The REA value chain is based on Porter’s definition. It is built by a number of value activities. A value activity takes some resources as input and creates some resources as output. From an economic perspective it is important that the output is considered to be of higher value than the input. On a high level of abstraction there are two ways to create additional value by an activity: firstly, one may use and/or consume some input resources in order to produce some output (e.g., a finished good)—this is called a transformation in REA. Secondly, in a trading relationship with external business partners one may receive resources (e.g., material, equipment, transport service, etc.) and give resources (e.g., cash) in return,—this is called a transfer in REA.
Furthermore, REA is built on the economic principle that any output by one value activity serves as input to another value activity. It follows that it is the resources which connect the different value activities. Thus, a REA value chain contains a number of value activities and specifies the resource flows amongst them—nothing more, nothing else [15]. More details are available on the second layer—the duality specification layer—where we find duality models for each of the value activities (cf. Figure 4).

The left hand side of Figure 3 presents the meta-model of the REA value chain. A value chain includes one to many value activities that are depicted by rectangles with rounded corners (cf. right hand side of Figure 3). A value activity is used only once in one distinctive value chain. A value activity points to exactly one duality (described in the next subsection). A duality is usually the basis of one value activity, but may be referred to by multiple value activities.

Resource flows tie the value activities together. A resource flow is a directed association that usually starts from a source value activity and ends at a target value activity (cf. right hand side of Figure 3). When analyzing a whole company, there is in theory no final output and no input that is not based on an output of another value activity. For the purpose of a partial analysis, we permit resource flows that have either no source value activity or no target value activity. It follows that a value activity has at least one, but up to many outgoing resource flows. Similarly, a value activity has at least one, but up to many incoming resource flows. Each resource flow points to exactly one resource. This resource is depicted by the symbol of a drop next to the directed arc of the information flow. A resource may be included in many resource flows. The right hand side of Figure 3 shows an abstract example model of a value chain which is a valid instance of the meta-model on the left hand side.

B. REA Duality

In the previous subsection, we learned that value activities receive some input resources to create output resources of higher value. Each value activity is further detailed by a duality on the second REA layer. A duality is a core economic principle that says that it is impossible to get something for nothing ("there is no free lunch"). Accordingly, a duality consists of two parts: the decrement entity set covers events executed by some agents leading to a decrease of some resources. It is compensated by the increment entity set that covers events executed by some agents leading to an increment of some (other) resources. By definition the increment in resources is considered of higher value than the decrement in resources. Again, the duality concept applies to transfers (exchanges with external agents) and transformations (value creation inside the enterprise).

Figure 4 shows the meta-model for a duality. The meta classes with white background describe the existing REA concepts, the ones with gray background represent our proposed extensions described further below. A duality has two specializations: a transfer and a transformation. Independent of the specialization a duality is composed of exactly one increment entity set and one decrement entity set. Both are specializations of the general entity set. Each entity set is represented in a specific swimlane (cf. Figure 5). According to the REA meta-model, an entity set covers at least one but up to multiple events. An event—depicted as a hexagon—is specific to the entity set it belongs to (cf. Figure 5). Following the principles of duality, all events in the decrement entity set (give/consume/use) are counterbalanced by the events in the corresponding increment entity set (take/produce) of the same duality (cf. Figure 4).

The relationship between an event and a resource is described by the concept of stockflow [15]. A stockflow is represented as a directed arc between exactly one event (hexagon) and one resource (drop) (cf. Figure 5). In the increment set the direction of the arc goes from the resource to the event, in the decrement set in the reverse direction. An event will affect most of the time one resource only, but it may affect multiple ones. Thus, an event may have one up to many stockflows connected. A resource usually is affected by many different events (in different entity sets of different duality models). At a minimum a resource is affected by one event—otherwise it would not be worth considering the resource at all. Consequently, a resource is connected to one up to many stockflows.

In REA, resources can be goods, material, rights, labor, equipment, physical assets, or services. REA does not make any particular differentiation and all of these resources are denoted by the icon of a drop (cf. Figure 5). Due to its dedicated focus on the production domain, ISA-95 differentiates between material, equipment, and physical asset as special kinds of resources. When aiming for an integrated approach the differentiation of these special resources should be reflected in the REA ontology as well. Accordingly, we define material, equipment, and physical asset as specializations of the REA

Authorized licensed use limited to: TU Wien Bibliothek. Downloaded on August 28,2021 at 13:32:38 UTC from IEEE Xplore. Restrictions apply.
C. REA task specification layer

In the first two subsections, we elaborated on the top two layers of REA (value chain specification layer and duality specification layer). One may expect that we do the same for the third layer—the task specification layer—describing the process to transform the input to the output as defined in the layers above. However, the REA literature does not concentrate on the task specification layer, instead it suggests to use activity diagrams or state machines to describe the task specification layer. REA does not provide any language concepts for linking identified tasks with agents, resources, etc. Accordingly, one may consider either extending the REA ontology for this purpose or specifying transformations to another language. In the context of the production domain, we are confident that ISA-95 is a perfect candidate language for the latter case. Accordingly, we propose that each REA duality model points to exactly one ISA-95 operations segment (see upper right corner of Figure 4). The relevant ISA-95 meta models with respect to an operations definition are described in the following section.

V. THE ISA-95 META MODEL

Production operations are defined by the ISA-95 operations definition model that is depicted in key parts in Figure 6. An operations definition represents the resources required to perform a specified operation. The operations definition references a work definition, which defines the information used to instruct a manufacturing operation (i.e., how to perform the operation) [18]. An operations definition is associated to one to many operations segments. Operations segments may be recursively structured. An operations segment encapsulates the information needed to quantify a segment for a specific operation. It corresponds to one to many process segments [18]. Process segments are the smallest elements of manufacturing activities that are visible to business processes.

An operations segment provides a logical grouping of personnel resources, equipment resources, physical asset resources, and material required to perform a specific operation. Consequently, it includes different kinds of resource specifications, personnel specifications, material specifications, equipment specifications, and physical asset specifications [4]. These resource specifications identify the...
resource types and/or concrete resources, their quantity and the unit of measure of the quantity needed to perform an operations segment. For instance, to perform a frame production, we need—amongst other things—a specified quantity of a certain material (e.g., carbon crossbar) or material type (e.g., crossbar). In addition, the operations segment may include one to many parameter specifications containing the names and types of values that may be sent to the manufacturing execution systems at Level 3 to parameterize an operation.

Each of the above mentioned resource specifications within an operations segment are references to corresponding ISA-95 models [4]. The material model contains the actual materials, material definitions, and information about classes of material definitions. Material information includes the inventory of raw, finished, intermediate materials, and consumables [18]. The role-based equipment model contains information about specific equipment, the equipment class, and their particular properties. Role-based means that the equipment model is used to construct hierarchy models used in manufacturing scenarios (enterprise, site, area, work center, work units, process cells, etc) [18]. Due to this role-based view the equipment model is related to the physical asset model [4]. This model contains information about the physical piece within the manufacturing enterprise, i.e., a specific equipment. The personnel model contains information about specific personnel (class Person), classes of personnel (class Personnel Class) as well as their properties [4].

Accordingly, the schemata of these resource models are very similar and we do not detail all of them due to space limitations. We pick the material model as a typical representative of the resource models and present it in Figure 7. A material class may be defined as containing an assembly of material classes and as part of an assembly of material classes. A material class is a grouping of material definitions for an operations definition. A material class may define zero or more material class properties. Material class properties may contain nested material class properties. These properties often list the nominal, or standard values for the material (e.g., pH factor, material strength). A material property does not have to match material class properties. A material definition shall belong to zero or more material classes. Similar to material class, a material definition may be defined as containing an assembly of material definitions and as part of an assembly of material definitions. For a detailed description of the ISA-95 resource models, we refer the interested reader to the standard [4].

VI. TRANSFORMATION RULES FROM REA TO ISA-95

In the previous two sections, we described the REA meta model and the relevant parts of the ISA-95 meta model. In our integrated modeling framework, we intend to use REA for the purpose of modeling the main business functions of an enterprise. These business functions reside on Level 4 of the functional hierarchy model as depicted in Figure 1. In REA, one may distinguish business functions that require the exchange of resources with business partners, i.e., REA-transfers, and business functions that require the transformation of resources and are executed within the enterprise, i.e., REA-transformations. In an industry context, the latter ones are typically the production processes. Evidently, information about business functions describing transformations should be passed to the control functions at Level 3 of the functional hierarchy model. Accordingly, the relevant information in REA models has to be transformed to ISA-95 in order to realize the upper part of our intended integrated modeling framework. In this section, we describe the corresponding transformation rules which are depicted in Figure 8.

Each REA duality describing a transformation is transformed to an ISA-95 operations segment (A). The name of the duality becomes the operations segment ID (A1). Alternatively, one may decide to use logical, system generated identifiers, in which case a REA duality ID would map to the operations segment ID and the name of the duality to the operations segment description. In this paper, we have opted for “readable” IDs, also for other concepts described further below. The REA duality also links to a corresponding process definition which is carried forward to the process segment ID referenced by the operations segment (A2). It should be noted that each REA duality model leads to exactly one operations segment. In case that this operations segment is not fine granular enough for control functions, one may re-work the operations segment in ISA-95 to create nested operations segments within it.

In the next steps, we have to transform the input resources for operations segments, which are personnel (B), equipment (C), physical assets (D), and materials (E). In REA the input side is described within the decrement entity set. Accordingly, calculating the input requires to access all events within the decrement entity set of a duality. The input then corresponds to the REA agents connected by participation associations to these events and the REA resources connected by stockflow associations.

It follows that each agent or agent type connected to a decrement event leads to a personnel specification within the operations segment (B). In the case of an agent type its name is mapped to the personnel class ID (B1). Whereas the name of a specific agent maps to the person ID of the personnel specification (B2). The participation association between an event and an agent has by default an attribute quantity, i.e. the number of agent (types) involved. This quantity is mapped to the personnel specification quantity (B3).

An equipment or equipment type connected to a decrement event results in an equipment specification as part of the operations segment (C). In case of an equipment type its name maps to the equipment class ID (C1). Whereas the name of a specific

Fig. 7. ISA-95: Material Model [IEC 62264-2]
Also the transformation rules for input materials (E) are similar to the ones for equipment (C) and physical assets (D). Evidently, the quantity of materials is not always a number of pieces. Consequently, there is an additional transformation rule mapping the unit of measure of the quantity of a stockflow to the unit of measure of the material specification (E4). However, most important is the fact that a material connected to a decrement event is considered as an input and thus the attribute material use of material specification is set to the value consumed (E5).

The transformation rules B - E describe the input side. The transformation rules for the output side are the ones in section F. The output of an operations segment is by definition the produced material or material type (including the specializations semi-finished goods and finished goods). In REA, the output are materials or material types connected via stockflow associations to events that reside in the increment partition. Accordingly, the transformation rules for output materials (F) are the same as for input materials (E) except for the fact that they apply to the increment side and not to the decrement side. In addition, the material use attribute is set to produced (F5). Furthermore, it is worth mentioning that semi-finished goods and finished goods are specializations of materials, and thus the transformation rules in sections E and F apply as well.

The transformation rules described above are used to map REA duality models to ISA-95 operations segments. In ISA-95, operations segments are not stand-alone items, but are always part of an operations definition. At first sight, one might assume that a REA value chain maps to a single operations definition and all duality models in the value chain become part of this operations definition. However, such an approach is too naive in practice. Our practical experience has shown that usually some duality models are grouped into one operations definition, but it always requires a human decision on this grouping. Accordingly, the transformation of a value chain to an operations definition is always a semi-automatic process requiring feedback from the modeler.

In this paper, we concentrated on the transformation of duality models to operations segments (of operations definition items), because they have a high significance for our approach. Nevertheless, it is important to note that REA also offers concepts to model the attributes of resources (equipment, physical assets, and material) and of agents as well as of their typification. The underlying meta model (cf. [11]) is conceptually very similar to corresponding ISA-95 resource models. Consequently, the transformation is rather straightforward and we do not further elaborate on them due to space limitations.

### VII. REA to ISA-95 Transformation Example

#### A. The REA model of Maxi Bike

The business model of Maxi Bike is to produce and sell bicycles. Figure 9 presents Maxi Bike’s value chain, which is an instantiation of the value chain meta model depicted on the left hand side of Figure 3. Keeping the example simple and easy to follow, we only present a partial analysis and do not show value activities for acquiring equipment, physical assets, raw materials and labor. The value chain covers five value activities: Purchase, Transport, and Sale

---

**Fig. 8. Transformation Rules for ISA-95 Operations Segments**

<table>
<thead>
<tr>
<th>Transformation Type</th>
<th>ISA-95 Operations Segment Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Duality ➔ Operations Segment ID</td>
</tr>
<tr>
<td>A1</td>
<td>Duality Name ➔ Operations Segment ID</td>
</tr>
<tr>
<td>A2</td>
<td>Duality ProcessDefinition ➔ Operations Segment ➔ ProcessSegment ID</td>
</tr>
</tbody>
</table>

Authorized licensed use limited to: TU Wien Bibliothek. Downloaded on August 26, 2021 at 13:32:38 UTC from IEEE Xplore. Restrictions apply.
are REA-transfers requiring horizontal integration, whereas Production and Assembly are REA-transformations requiring vertical integration. The value chain shows the flow of resources (materials/equipments/physical assets) amongst them. In Purchase the resource Cash is used to get the material type Wheel. Wheel and again Cash are used in Transport to receive the wheels at the right location (Production Unit B).

The ingoing resource flows of the value activity Frame_Production are the material type Foot Pedale, the material 25CrMo4::Crossbar, the equipment Production Unit A and the physical asset types Assembly Jig and Bending Machine. The outgoing resource flow of this value activity is the specialized material F1100::Bicycle Frame which is a semi-finished product. The value activity Assembly has as ingoing resource flows the material types Seat, Screw, Wheel and as semi-finished product F1100::Bicycle Frame. The other ingoing resource flows are the physical asset type Screwdriver and the equipment Production Unit B. These resources are transformed (i.e., used and consumed) to produce the specialized material BY1100::Bicycle which is the finished product. In the value activity Sales the BY1100::Bicycle is turned into Cash which is used as input for the other value activities mentioned above.

Each of the five value activities presented in Figure 9 must be refined by a duality model. Due to space limitations we only show the duality model for Frame_Production and Assembly (cf. Figure 10). The right hand side of Figure 10 depicts the duality model Assembly which is again a Transformation. The compose_in: decrement event is performed by the inside agent Joe::Assembler and leads to a decrease of the input resources by consuming the semi-finished product F1100::Bicycle Frame and the material types Wheel with a quantity of 1, Screw with a quantity of 15 and Seat with a quantity of 1. In addition, the physical asset type Screwdriver (quantity 1) and the equipment Production Unit B are used. This decrement event is compensated by the increment event compose_out, which produces the specialized material BY1100::Bicycle as final product received by the agent type Product Manager. These example models do not specify any process details on how to produce the bicycle frame or assemble the bicycle. They only provide links to the Process Definitions FTP1 and FA2.

B. Mapping the REA Duality Models to B2MML

In contrary to the REA-DLS, ISA-95 does not come with any dedicated graphical language as a concrete syntax to represent ISA-95 compliant models. Accordingly, one may only use a corresponding object diagram as an abstract syntax. However, the Business To Manufacturing Markup Language (B2MML) [19] is an XML implementation of ISA-95. In other words, B2MML defines XML schemas that are exact equivalents of the ISA-95 meta model. Accordingly, one may use a B2MML XML file that is valid with respect to the B2MML schema to show a valid instance of the ISA-95 standard. This is our choice for illustrating the example.

In the following, we demonstrate the mapping of the two duality models Frame_Production and Assembly as depicted in Figure 10 to B2MML. This mapping uses the transformation rules of Figure 8. The resulting B2MML file is listed in Figure 11. For easier readability, we do not use closing XML tags, but use indent style instead. This B2MML file lists an operations definition with two operations segments (Frame_Production and Assembly), which are both one to one mappings of the REA duality models Frame_Production and Assembly. All the information in green font is a result of applying our transformation rules. It should be noted that the grouping of the two duality models or operations segments, respectively, has been done manually and, consequently, the instances in black font have to be created manually.
The first operations segment presented in the B2MML example in Figure 11 with the ID Frame_Production is a one to one mapping to the REA duality model Frame_Production. This operations segment contains a process segment ID BY1100, that corresponds to the link specified in the REA duality model. The personnel specification of the operations segment contains the personnel class ID Construction Engineer who participates in the decrement event build_in attributed by a quantity of 1. The equipment specification with its ID Production Unit A and the physical assets specifications Assembly Jig and Bending Machine are mapped according to the equipment and physical asset types connected to the build_in decrement event. Each of them has a quantity of 1.

The decrement event build_in expects a material type Foot Pedale and a material 25CrMo4 which is of material type Crossbar. Accordingly, we have two material specifications. The first one is for the material class ID Crossbar and the exact material definition ID 25CrMo4, whereas the second one only mentions the material class ID Foot Pedale without any more detailed material definition. These material specifications are considered as input resources and thus the attribute material use, of both of them, is set to the value Consumed. The material class ID Bicycle Frame with the material definition ID F1100 has the status Produced with a quantity of 1, which is a mapping result of the increment event build_out. The transformation of the duality Assembly to the second operations segment is done in the exactly same manner, and thus, is not described in further detail.

VIII. CONCLUSION

It is our overall goal to develop a universal model-driven approach towards the horizontal and vertical integration in the context of smart factories. For this purpose we strive for an integrated modeling framework based on existing modeling approaches. Thereby, we build up on the REA business ontology to identify, both, activities requiring horizontal integration with business partners and activities serving as hooks into the internal systems requiring vertical integration. The latter activities have then to be further detailed by means of the ISA-95 standard. Accordingly, it is of crucial importance to transform concepts of REA to concepts of ISA-95.

First of all, this requires an alignment of concepts that appear to be similar in REA and ISA-95. In this respect, we have extended the resource concept in REA by similar concepts from ISA-95. In particular, we introduce specializations of the concept resource, namely equipment, physical asset, and material. Evidently, these extensions also apply to the REA type level.

Most importantly, we have developed dedicated transformation rules for the purpose of transforming a REA model into an ISA-95 one. In particular, we map REA duality models to ISA-95 operations segments. Thereby, we are able to convert information about the input and output of business functions to the control functions. Nevertheless, it is important to note that later on this information needs to be further detailed on the shop floor control level.
For the evaluation of our approach, we have first implemented the proposed REA extensions into our REA DSL tool. In a next step, we added the transformation rules to our tool. For the moment these rules have been hard coded, but it is planned to use a dedicated transformation language in the future. Accordingly, we demonstrated the technical feasibility of our approach by mapping from REA-DSL to B2MML (the XML equivalent of ISA-95). The syntactical correctness of the transformation has been checked by the proof of valid B2MML XML instances. More extensive case studies are planned for the future, once the overall modeling framework spanning over all hierarchical layers has been realized.

REFERENCES


3 AutomationML, ISA-95 and Others: Rendezvous in the OPC UA Universe

AutomaticML, ISA-95 and Others:
Rendezvous in the OPC UA Universe

Bernhard Wally1, Christian Huemer2, Alexandra Mazak1 and Manuel Wimmer2

Abstract—OPC Unified Architecture (UA) is a powerful and versatile platform for hosting information from a large variety of domains. In some cases, the domain-specific information models provide overlapping information, such as (i) different views on a specific entity or (ii) different levels of detail of a single entity. Emerging from a multi-disciplinary engineering process, these different views can stem from various tools that have been used to deal with that entity, or from different stages in an engineering process, e.g., from requirements engineering over system design and implementation to operations. In this work, we provide a concise but expressive set of OPC UA reference types that unobtrusively allow the persistent instantiation of additional knowledge with respect to relations between OPC UA nodes. We will show the application of these reference types on the basis of a rendezvous of AutomationML and ISA-95 in an OPC UA server.

I. INTRODUCTION

OPC UA is seen as the future of communication in industrial settings, and has been designed in a very flexible way in order to be able to adapt to future demands and developments [1]. As such, domain-specific information has been kept out of the standard as strictly as possible [2]. Instead companion specifications are to be developed for introducing meaning (semantics) to the syntax provided by OPC UA. This is also a necessity in order to foster interoperability of tools in industrial automation. Interestingly, due to the flexibility of OPC UA, an OPC UA server, which is the host for OPC UA information models, can be "polluted" with overlapping and partly redundant (possibly contradicting) information from a variety of sources and domains.

In this work we will examine what kind of overlapping information is typically available in OPC UA servers by investigating on a set of OPC UA companion specifications and how this overlapping information can be aligned in a way that IT systems can deal with this kind of information more confidently.

The paper is structured as follows: after some relevant and necessary background information (Sec. II), we will discuss related work (Sec. III) before we explicate and evaluate our approach (Sec. IV). Following a critical discussion (Sec. V) we conclude and provide further research directions (Sec. VI).

A. OPC UA

OPC Unified Architecture (UA) is a series of standards brought forward by the OPC Foundation; it defines a modern, object- and service-oriented communication stack and modeling paradigm for industrial automation [2]. Its versatility allows the mapping of various domain-specific models into OPC UA equivalent information models, such as AutomationML and ISA-95 models. OPC UA implements a type-object pattern [3], [4], [5] which enables domain modeling (i) at runtime and (ii) decoupled from the OPC UA core standard.

B. AutomationML

The Automation Markup Language (AutomationML) is based on Computer Aided Engineering Exchange (CAEX), which is a data format that has been defined in the scope of IEC 62424 and provides structures (i) for information exchange between piping and instrumentations diagram tools and process control engineering related computer aided engineering tools, as well as (ii) for the representation of process control engineering requests in piping and instrumentations diagrams [6]. CAEX is based on XML and enables the metamodeling and modeling of, e.g., the hierarchical architecture of a plant, including involved machines and controllers and their physical and logical connections.

AutomationML is standardized as IEC 62714 and defines sets of role classes and interface classes with certain restrictions regarding their application [7], [8]. AutomationML defines an abstract interface class ExternalDataConnector which is used to reference external documents and elements therein. Two use cases of this external data connector have been defined so far in separate whitepapers: (i) COLLADAInterface specifies how external COLLADA3 documents are referenced [9] and (ii) PLCopenXMLInterface defines how PLCopen XML documents (which are based on IEC 61131-3 [11]) can be referenced from AutomationML documents [12]. This referencing mechanism plays an important role in the analysis presented later.

1Bernhard Wally, Alexandra Mazak and Manuel Wimmer are with the Christian Doppler Laboratory for Model-Integrated Smart Production, Institute of Information Systems Engineering, TU Wien, 1040 Vienna, Austria, {wally, mazak, wimmer}@big.tuwien.ac.at
2Christian Huemer is with the Business Informatics Group, Institute of Information Systems Engineering, TU Wien, 1040 Vienna, Austria, huemer@big.tuwien.ac.at
3COLLADA—Collaborative Design Activity: an XML based exchange format for 3D assets (cf. https://www.khronos.org/collada/)
4PLCopen is a vendor- and product-independent association active in industrial control (cf. http://www.plcopen.org); PLCopen XML is a data exchange format for the storage of programmable logic controller (PLC) program information according to IEC 61131-3 [10]
C. ISA-95

ISA-95 (IEC 62264) is a series of standards that addresses the integration of the enterprise domain with the manufacturing and control domains. It defines a set of object models for the exchanging of information between these domains—it provides a standard terminology and set of concepts for system integration [13]. The relevant part of ISA-95 for this work is part 2, specified in IEC 62264-2:2013 [14], which defines common objects and attributes such as personnel, equipment, material, process segments, schedules and performance data. An XML serialization of ISA-95 has been defined in [15] under the name “business to manufacturing markup language” (B2MML).

III. RELATED WORK

One of the earliest OPC UA companion specifications was about the modeling of ISA-95 data: in [16] an initial OPC UA information model is provided that defines the base entities from which domain-specific OPC UA nodes must be derived in order to be valid ISA-95 instances. This specification is one of the foundations for our analysis.

Similarly, in [17] a first draft for the modeling of AutomationML information in OPC UA was presented, that was later standardized in [18]. There, the rules for transforming AutomationML entities into OPC UA entities are defined alongside an initial OPC UA information model that is to be used as a base for own AutomationML models. The standardized specification is another important input to our analysis presented in Sec. IV.

An AutomationML application recommendation is prepared in [19], where the alignment of AutomationML and ISA-95 is described, i.e., how to encode ISA-95 related information in AutomationML documents or how to refer to external ISA-95 information stored in separate files. This application recommendation is the third main pillar for the analysis given in this work.

In [20] it is shown that vertical integration can go beyond the integration of just two domains. There, besides AutomationML and ISA-95, even an accounting/business metamodel is brought into the vertical integration chain: the Resource-Event-Agent business model language [21]. The approach presented in this work further strengthens the knowledge-base of manufacturing enterprises and could be part of the enabling technologies for reaching from the business model down to the shop floor.

An initial discussion of the alignment of AutomationML and ISA-95 is given in [22], and a bit more in-depth in [23]. However, the approach that has been presented in [23] imposes workarounds for specific instance constellations (e.g., multi-classification of equipment), that could be solved in a more elegant way by changing some of the metamodel mappings that have been defined there. Specifically, equipment classes should be better modeled in terms of AutomationML role classes to overcome the single-classification restriction given by the utilization of system unit classes. Such improvements have been already incorporated in [19].

A production data transformation scheme is brought up in [24], targeting faster data processing for, e.g., decision support systems. There, ID based lookup of entities from a storage system is employed for data retrieval. The mechanism presented in this work would enable a different lookup mechanism based on explicit linking of related entities.

Aligning enterprise information systems (EIS) with production systems is described in [25], using a service oriented approach. Given, that entities of both the business and the production domain are available in OPC UA, this is another use-case for the approach presented in this work. A related idea is superficially presented in [26]: using formalism from ISA-95 in order to refine coarser-grained concepts from an EIS, such as those emerging from REA (cf. also [27] for the applicability of REA as an EIS). Also there, explicit linking of entities within a dedicated communication protocol would help superordinate systems in their orchestration tasks.

The work presented here can be related to the topic of model composition, which is tackled in [28] and that defines an abstract model-weaving metamodel enabling the specification of relations such as “rename”, “override” and “merge”. For us, this approach is one step too generic, as we would like to explicitly name the kind of inter-model relations, and make this set of relations fixed (especially from a semantics point of view).

Explicit coupling of corresponding elements in models of different domains is presented in [29]. It makes use of a dedicated “linking” metamodel that is used to describe the kind of relationship between corresponding elements. This approach seems to be very promising, as it does not impose changes to the domain meta-models. It could serve as a base for specifying similar behavior in the realm of OPC UA information models. In fact, we are building on the findings discussed there.

A different approach was taken in [30], where AutomationML was transformed into a Resource Description Framework (RDF) representation, which is a generic ontology modeling language. Linking of semantically related entities is realized in this ontological domain, and standard ontology-tools can be used for querying and browsing the loaded AutomationML model(s). It might be worthwhile to use their toolset for the reasoning about inter- and intra-modal relations and then convert them to OPC UA references in a live information model, as we do it in the approach presented in our work.

Inter-model dependencies in production system engineering comprising software, electrics and mechanics are presented in [31]. There, the importance of understanding models of different domains as different views on the same underlying physical aspects is made clear. Their approach was based on the systems modeling language (SysML) and leveraged internal block diagrams and their ports and flows concepts for modeling different engineering views. In our example, SysML would be yet another domain model that might be translated into OPC UA at one point and that would be linked to other domain models by the references defined in our work.
A slightly more complex approach is presented in [32], describing an implementation concept for the “administration shell” of Industry 4.0 components [33]. There, multiple OPC UA servers are employed and they interact with each other through dedicated OPC UA clients. The approach presented in our work would enable a semantic linking of components in different OPC UA servers, since OPC UA references are allowed to point to nodes that are deployed in another OPC UA server [1]. This semantic knowledge could in turn be used for the (auto-)configuration of OPC UA clients dealing with inter-server communication.

IV. RENDEZVOUS IN THE OPC UA UNIVERSE

A. Specific Problem Domain

Overlapping domain-specific models not only exist in the case of AutomationML and ISA-95, but for example with AutomationML and PLCopen. In [34], an information model definition is given that enables the transformation of PLCopen information into an OPC UA information model. Additionally, in [12], a recommendation for the inclusion of PLCopen information into AutomationML documents is given, based on PLCopen XML [10]. So also in this case, two different, yet interwoven domains can be both converted into OPC UA information models by separate means, i.e., there exist rules for transforming each of the domain models into OPC UA, but each of these rule-sets is not necessarily taking the other rule-sets into account.

In [35], a typical plant engineering process is depicted showing the different steps in design, planning and operation. In a fully digitized workflow it is very likely that some of the modeled artifacts are available in an OPC UA server, many of which originating from different tools following different metamodels, implementing diverse domain knowledge. Deploying such artifacts in parallel with each other requires some kind of methods to formulate their relationship to each other (e.g., how about the relation between a physical production machinery and its digital twin?).

It is infeasible to make all OPC UA mapping standards (e.g., [36], [37], [38]) aware of each other and harmonize their information models as far as possible, because (i) the amount of bilateral reconciliations increases dramatically with the addition of another standard and (ii) the standards would have to be edited quite often which would render them inconvenient standards (which are usually supposed to be rather stable). Fig. 1 shows an excerpt of the domain-specific base information models for AutomationML and ISA-95 within OPC UA. It can be observed that the domain-specific object types form two disjoint sub-trees. The consequence is that user-defined object types of an application specific information model can not be modeled as sub-type of both domains. This is due to OPC UA modeling rules that allow only single-inheritance for object type hierarchies.

Instead we believe that this problem would be better addressed at a higher level of abstraction, and it could follow the approach given in [29] by making inter-model-relationships explicit. This could be realized by a separate information model or it could be integrated into the OPC UA standard information model as a means for modeling such relations.

To make our point more clear, let’s consider the example of AutomationML and ISA-95 again: it would be very elegant to be able to provide the functionality depicted in Fig. 2. The mapping relations resemble transformation rules for the conversion of information from one domain into information of another domain: (i) function $f$ describes how to transform AutomationML models into the OPC UA space, (ii) function $h$ describes the transformation from ISA-95 to OPC UA, and (iii) function $g$ provides a transformation from ISA-95 to AutomationML. It would be beneficial for a number of use cases if the function composition $f\circ g = h$ would hold, i.e., ISA-95 models that are mapped directly into the OPC UA space via function $h$ have an identical representation there as if they would first be transformed into AutomationML (via $g$) and only then be mapped into the OPC UA space (via $f$).

However, it is not possible to provide this function composition, as we can easily show with an example. Consider an AutomationML internal element that implements the ISA-95 role class Equipment. It would be instantiated as an OPC UA object with a type definition pointing to a sub-type of the OPC UA object type AutomationMLBaseRole following [18]. An equipment instance of ISA-95 would in turn be instantiated as an OPC UA object with a type definition pointing to the OPC UA object type EquipmentType following [16]. Since an OPC UA object is allowed only one type definition, it is not possible to create an OPC UA object type that is a sub-type of both AutomationMLBaseRole and EquipmentType. Consequently, the transformation rules will create two disjoint object type sub-hierarchies that can be connected with each other using OPC UA references.

A first approach is given in [18], where an OPC UA reference type HasAMLUAReference is defined. However,
its use is tailored to specific inter-model-relationships that are typical for AutomationML documents. This reference type is a directly usable sub-type of the standard OPC UA reference type NonHierarchicalReferences. It is used to refer from an OPC UA node to stems from an AutomationML model to an AutomationML model of another domain. Its main use case is in the automatic conversion of ExternalDataConnectors that may occur within AutomationML documents and that refer to external data that is stored in separate non-AutomationML files. Fig. 3 depicts an AutomationML document that refers via a specialization of the external data connector, the B2mmlReference, to a B2MML document containing ISA-95 information. A transformation of this AutomationML document to OPC UA would generate (at the bottom of the information model) a set of AutomationML nodes, and (at the top of the information model) a set of ISA-95 nodes. The B2mmlReference would be transformed as a HasAMLUAReference reference pointing from the AutomationML node to the corresponding ISA-95 node.

Fig. 3. OPC UA nodes of different domains deployed in a OPC UA server, and an explicit relation between them.

The semantics of HasAMLUAReference is simple: there is a relation from an AutomationML node to a non-AutomationML node [18]. This loose definition imposes rather weak application constraints that allow for great flexibility of this reference type, but at the same time provide only little additional knowledge about the kind of relation between nodes that participate in such a relation. Also, it is on one side restricted to the AutomationML domain. Therefore, we propose a set of reference types that allow the modeling of more precise semantics in OPC UA servers that contain overlapping multi-domain nodes.

B. Explicit References

In order to relate nodes of different domains to each other, various relation come into one’s mind: equivalence, refinement, part-of, specialization-of, etc. Inter- and intra-model relations in the context of requirements engineering have been materialized in [39]. As the intra-model relations should be normally supported by the corresponding domain model, we omit them and find the following four inter-model relations: refines, satisfies, tracedFrom and verifies (tracedFrom is a very generic relation that has no clear semantics and is left out in our work). In [29], the following relations have been defined: refines, equivalent-to and satisfies. In their work, they have been using these relations for dealing with inconsistencies between different domain models. This is not the focus of this work, where we want to establish relations between potentially consistent models, however, as we will see the set of references is very similar; they same is valid with the previously mentioned relations defined in [39]. Finally, the following relations are identified (cf. Fig. 4):

1. RepresentsDifferentView This symmetric relation expresses that two nodes represent the same (logical or physical) entity, but in different levels of detail. It is a sub-type of NonHierarchicalReferences. Application scenarios for this reference type include: (i) a node representing a static design model related to a node representing a live object that is continuously updated at runtime, (ii) a node representing a robot from a plant planning perspective related to the same robot from a “behavior-teaching” perspective, (iii) a node representing a product from a sales perspective related to the same product from a production perspective, (iv) a digital twin related to its physical counterpart.

2. HasRefinement This asymmetric relation expresses that two nodes represent the same (logical or physical) entity, but in different levels of detail. It is a sub-type of RepresentsDifferentView. Its inverse relation is named IsRefinementOf. An application scenario for this reference type is a node representing a person modeled in AutomationML related to a node representing the same person, but modeled in ISA-95 with much more details, e.g., corresponding personnel classes and their qualification test specification, the qualification test results of this person, etc.

3. HasImplementation This asymmetric relation expresses that one node verifies another one, such as (i) a test case verifying a requirement or (ii) operations data verifying the accuracy of a simulation model. It is a sub-type of NonHierarchicalReferences. Its inverse relation is named IsImplementationOf.

4. HasVerification This asymmetric relation expresses that one entity represents an implementation of another entity. It is a sub-type of NonHierarchicalReferences. Its inverse relation is named IsVerificationOf.
C. Evaluation

We are evaluating our set of reference types on the following set of domain models: AutomationML, ISA-95 and MTConnect [40], as depicted in Fig. 5. AutomationML is typically used to model the equipment used in manufacturing environments, such as the machinery/robots/tools, controllers and communication links. ISA-95 provides facilities for modeling the same kind of information, but it is not as flexible in its modeling as AutomationML; but it allows modeling explicit domain knowledge such as processes, and production steps, as well as runtime data such as schedules, performance data and capability/capacity information.

MTConnect is a communication protocol for the shop floor based on HTTP and XML. It is trimmed for communication with machine tools and provides a rich vocabulary for this kind of equipment.

For each of these domain models there exists a mapping to OPC UA, or they are natively designed in OPC UA. Typically, an MTConnect-aware machine tool such as a milling device would be modeled in an OPC UA information model three times: (i) from the MTConnect perspective, (ii) as an AutomationML system unit class and (iii) as an ISA-95 physical asset class. Usually, modeling conventions such as naming policies or specific attributes and their values would be needed in order to allow a reasoner to find out that a specific node has corresponding nodes hanging out in the same OPC UA space. However, by using explicit links, this knowledge can be persisted and used by other tools which are not introduced to the modeling conventions in use.

Fig. 5 depicts a milling device named MyMilling that is represented three times in the OPC UA server: as an AutomationML internal element, as an ISA-95 physical asset and as an MTConnect device. Using the RepresentsDifferentView reference type, it can be made clear, that the different OPC UA nodes are really all describing the same physical entity, but from their domain specific perspective. A similar example is given with the entity MyWorker that is modeled in AutomationML only as a stub element (usually personnel is not modeled in AutomationML), but modeled in more detail in ISA-95. This refinement is expressed using a HasRefinement relation.

It turns out that there are use cases in which such explicit links make sense even within a single domain, e.g., if the domain model does not provide a facility to describe the relations discussed in this work. Consider a manufacturing company that requires a very specific machine “MyMachine” in order to produce their products. Imagine, that this machine is not readily available but needs to be tailor-made and that this manufacturer has its own workshop in which it can produce such machines. Given that ISA-95 is used to model this machine, it will need to be instantiated two times: (i) when it is produced by the internal workshop, it is an entity of the material information model, but (ii) as soon as it is used for producing the products of the company, it becomes an entity of the physical asset information model, implementing a specific equipment role. ISA-95 does not provide modeling support for this circumstance, but our RepresentsDifferentView reference type can be used to implement this relation (cf. Fig. 6).

V. CRITICAL DISCUSSION

It is possible (and likely) that the presented set of OPC UA reference types is not enough for specific application scenarios. In such a case new reference types need to be introduced that are related to the reference types presented in this work, or they could be independent from them. In the use cases that we have explored in the evaluation, the relations that have been defined here were sufficient to model general inter- and intra-model relations that provide additional knowledge that proofs useful for OPC UA client applications.

The example given in Fig. 5 exemplifies the usefulness of our approach: the reference type HasAMLUAReference that is defined in [18] relates two nodes with each other on the basis of an ExternalDataConnector. However, due to the modeling constraints imposed by [41], an AutomationML entity needs to create a child internal element representing a document that in turn defines an external interface that resembles the relation to the external entity. The result of this regulation is, that the HasAMLUAReference relates the document entity with the external entity. In some cases (e.g., in the cases described in [19]) the external reference depicts a correspondence of the parent internal element of the document with the external entity. This correspondence can be modeled in terms of a RepresentsDifferentView reference between the structurally and semantically better matching OPC UA nodes.

According to this, we believe that the proposed reference types may provide additional information to an observer by resembling semantically enriched hyperlinks between corresponding OPC UA nodes.

VI. CONCLUSION

We have presented an OPC UA node set comprising domain-agnostic reference types that can be used to model inter- and intra-model relations. The goal of our work is to make explicit some rather generic types of information in order to reach a more complete picture of the model(s) under observation. In the context of OPC UA, models of multiple engineering domains can emerge in the melting pot that is an OPC UA server. It is very useful to provide an understanding of close relations between entities in order to
Fig. 5. OPC UA information model of a milling device (MyMilling*) and a worker, seen from 3 (or 2 in the case of MyWorker) different views. Entities are suffixed with an abbreviation of the corresponding metamodel class (IE: internal element, SU: system unit, PA: physical asset, P: person, D: device). Some standard relations have been left out to make the diagram less cluttered.

allow automated tools or tools that require a human-in-the-loop to, e.g., unerringly select the correct view of an entity to execute an action, generate a report or display data in a user interface: the HasImplementation reference type could, e.g., be used to navigate between planning/design artifacts and implementation/operations artifacts.

Future work will further investigate application scenarios for the reference types defined in this work and strive for either extending or stabilizing the set of relations. One specific topic of interest is the linking of shop floor entities to business objects in order to foster vertical tool integration. Another stream of research might look into the interplay of multiple OPC UA servers.

ACKNOWLEDGMENT

The financial support by the Austrian Federal Ministry for Digital and Economic Affairs and the National Foundation for Research, Technology and Development is gratefully acknowledged.

REFERENCES


Authorized licensed use limited to: TU Wien Bibliothek. Downloaded on August 28,2021 at 13:31:48 UTC from IEEE Xplore. Restrictions apply.
OPC UA Standard

BaseObjectType

ISA95ObjectOfType

MaterialLotType

PhysicalAssetType

MyMachineMat

MyMachinePA

Fig. 6. OPC UA information model of a specific machine “MyMachinePA” (a physical asset required in the production process) that was internally produced as material lot “MyMachineMat”.


4 Leveraging integration facades for model-based tool interoperability

Leveraging integration facades for model-based tool interoperability

Galina Paskaleva,1,2, Alexandra Mazak-Huemer1,2, Manuel Wimmer1,3, Thomas Bednar1,4

1 Subsurface Engineering - Digital Transformation in Tunneling, Montanuniversitaet Leoben, Erzherzog-Johann Straße 3, 8700 Leoben, Austria
2 Institute of Business Informatics - Software Engineering, Johannes Kepler Universität Linz, Altenberger Straße 69, Science Park 3, 4040 Linz, Austria
3 Research Unit of Building Physics, Institute of Material Technology, Building Physics and Building Ecology, Faculty of Civil Engineering, TU Wien, Karlplatz 13/207/2, 1040 Vienna, Austria

ABSTRACT

Data exchange and management methods are of paramount importance in areas as complex as the Architecture, Engineering and Construction industries and Facility Management. For example, Big Open BIM requires seamless information flow among an arbitrary number of applications. The backbone of such information flow is a robust integration, whose tasks include overcoming technological as well as semantic and pragmatic gaps and conflicts both within and between data models. In this work, we introduce a method for integrating the pragmatics at design-time and the semantics of independent applications at run-time into so-called ‘integration facades’. We utilize Model-driven Engineering for the automatic discovery of functionalities and data models, and for finding a user-guided consensus. We present a case study involving the domains of architecture, building physics and structural engineering for evaluating our approach in object-oriented as well as data-oriented programming environments. The results produce, for such scenario, a single integration facade that acts as a single source of truth in the data exchange process.

ARTICLE INFO

Keywords
MDE
Data exchange
Semantics
Integration
Pragmatic integration
Heterogeneity

1. Introduction

Today, the integration of distributed, autonomous and heterogeneous data sources across application boundaries is gaining importance due to the increasing networking of organizations and companies in many application fields. This leads to the situation that the information flow goes hand in hand with the translation among different data models with different syntax and semantics. A current example for this evolution is the domain of Architecture, Engineering and Construction (AEC) as well as Facility Management (FM) industries. In these industries’ daily business, the volume of data to be exchanged among various stakeholders (e.g., building developers, energy suppliers, architects, structural engineers, building physicists, etc.) is rapidly increasing. Each of these stakeholders possesses specific domain knowledge and has a specific view of the project. These different perspectives may cause different forms of heterogeneity in the definition and handling of data, which hinders an interference-free communication within a building project.

This heterogeneity can be syntactic, semantic, pragmatic or a difference in the level of detail, i.e., granularity. Syntactic heterogeneity is caused by working with different data models in different tools. One example of semantic heterogeneity is the concept of homogeneous unit in the description of the building ground in different norms and regulations. The Austrian guideline defines it on the basis of homogeneous substrate, the Swiss norm on the basis of similar structural behavior. Pragmatic heterogeneity occurs, for instance, when in the absence of a clear regulation, one architect considers slabs with inclination of more than 15% as walls and another with inclination of more than 25%. An example of heterogeneity based on diverging levels of detail can be found in the following scenario. The architecture domain relies on geometry not...
only as a representation of a building project, but also as an information carrier, whereas the building physics domain is concerned with thermal and hygric flux through the construction of a building. For this reason, building physicists consider points of interest (e.g., the joints between walls) in much greater detail than architects do.

In addition to data heterogeneities, the data exchange within or across different phases of a project is challenging, since there are different exchange standards used as well as various requirements that must be considered. To overcome these obstacles, Building Information Modeling (BIM) has become more and more established in recent years. The main motivation behind BIM is to accommodate the heterogeneous nature of the AEC industries and involved domains, and to provide seamless data flows within any building or infrastructure project. To put it in a nutshell, BIM aims to support the generation and management of digital representations of physical and functional characteristics of a built structure throughout all phases. The ultimate goal of this development is Big Open BIM, a method for loss- and distortion-free, possibly real-time, data exchange across technological spaces and domains [7].

Preliminaries to BIM: The realization of BIM is not limited to a single data exchange standard. In fact, there have been multiple attempts to develop a suitable standard for it. For example, the proprietary Drawing Exchange Format (DXF) standard was widely used in the 1980s and 1990s, and is currently still in use. Furthermore, the open Standard for the Exchange of Product model data (STEP) was developed in the 1980s and, subsequently, became an ISO standard (ISO 10303). The Collaborative Design Activity (COLLADA) standard includes the exchange of geometric constraints and animations and is used not just in the AEC industries, but also in the animated production industry as part of the industry standard Automation ML [1]. The Construction Operations Building Information Exchange (COBie) was developed by the US military in 2007 and aims to store construction related data during all phases of a building’s life cycle. However, the currently most widely used BIM data exchange standard is the Industry Foundation Classes (IFC) standard [6]. The aim of IFC is to support Big Open BIM by providing a seamless communication among all stakeholders within the AEC domain. Therefore, it offers very abstract (e.g., IfcObjectDefinition) as well as very specific (e.g., IfcBoiler) concepts to describe a domain of interest. A deep look at IFC: Currently, IFC provides semantic types for the following domain groups: Building Controls, Plumbing and Fire Protection, Structural Elements, Structural Analysis, Heating Ventilation Air Conditioning (HVAC), Electrical, Architecture, and Construction Management [6]. Building physics, which we mentioned above, is not regarded as a domain, but as a resource. It is partially covered by module 8.10 Material Resource. Other sub-domains of building physics, e.g., acoustics, have not been integrated yet. Attempts to extend the standard or to define appropriate property and quantity sets for energy simulation tools have already been made, for example, by Chen et al. in [9] and by Bracht et al. in [3]. Similarly, domains involving underground facilities, such as tunnels, are yet to be fully developed [2,29]. Even the domains with good coverage cannot be regarded as semantically complete, because new technologies and methods enter the industry at a pace the IFC development cycle cannot keep up with. For example, the HVAC semantic types do not contain elements for thermal mass activation as a method directive, e.g., for floor heating. These examples outline only some of the major challenges even a widely adopted and rigorously developed BIM standard, such as IFC, has to face on the road to Big Open BIM.

BIM Tools: There are software tools developed specifically for a particular domain (e.g., C.A.T.S., DDS-CAD, Autodesk AutoCAD Mechanical [13] and Electrical [17] for the building services engineering field, Dlubal RFEM, Tekla Structures [16] and AUTOFORM [8] for the structural engineering field), or for multiple domains (e.g., Autodesk REVIT, Nemetschek Allplan and ArchiCAD). Each of these listed tools has its own, in most cases, closed data model. It is to be noted that these are only some of the most widely used software tools. In addition to them, there are many more, most of them dedicated to performing only a small subset of tasks within a single domain or project. However, in order to be “BIM-ready”, each and every software tool needs to implement some form of BIM.

1.1. Problem statement

Due to its very active development, wide usage and continuing efforts to integrate more and more domains, IFC presents a particularly suitable example for exploring the still existing technical challenges and missing links towards a realization of Big Open BIM via data exchange standards that honor the multiple types of heterogeneity we mentioned above. In the following, we outline three technical challenges and some practical examples as an illustration of their practical implications (see also Table 1):

Table 1 Problem statement outline: types of heterogeneity that require integration.

<table>
<thead>
<tr>
<th>No</th>
<th>Heterogeneity</th>
<th>Features</th>
<th>Issues</th>
<th>Integration implications</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Semantic</td>
<td>Semantic discrepancy between domains or regulations</td>
<td>Can contain explicit and implicit, formal and informal specifications</td>
<td>The implicit and informal parts can lead to errors</td>
</tr>
<tr>
<td>2</td>
<td>Procedural</td>
<td>Implicit assumptions</td>
<td>Cannot be included in an integration specification</td>
<td>Lead to errors</td>
</tr>
<tr>
<td>3</td>
<td>Syntactic</td>
<td>Heterogeneity between standards, seldom within the same field</td>
<td>Need for consistency checks</td>
<td>Easiest to automate, least potential to produce errors</td>
</tr>
<tr>
<td>4</td>
<td>Semantic modeling</td>
<td>Interlocking of syntax and semantics due to defects in the tools for semantic modeling</td>
<td>The semantics becomes syntax-dependent</td>
<td>A change of syntax during data exchange can lead to a hidden change in semantics</td>
</tr>
</tbody>
</table>

9 https://www.dds-cad.net/ (last accessed 2020-11-13).
There are multiple practical implications arising from the three challenges listed above.

(a) **Implementation overhead:** The manual translation to and from a standard as semantically complex as IFC is challenging. In practice, for small pieces of software, developed from scratch to perform a single dedicated task (e.g., a thermal flow simulation tool in the building physics domain), there are simply not sufficient resources (e.g., finances, person hours) for such implementations. This leads to a large number of small tools offering very similar, often state-of-the-art, functionalities that are never used beyond the limits of one project. This results in the loss of domain expert knowledge for the AEC communities.

(b) **No real-time feedback:** Communication via data exchange standards often involves serialization of large amounts of data, which does not allow real-time feedback. The delay between user action and observable results is not measured in milliseconds, but in minutes or hours. For instance, this makes the fine-tuning of a building simulation extremely tedious and time-consuming.

(c) **No single source of truth:** The implicit heterogeneity we described above produces divergence in semantics between implementations of the same standard. Therefore, instead of a single “source of truth” there are multiple competing ones. This invariably results in translation errors when transferring information from one implementation or model to another [7,20], even within the same software family, e.g., exchanging walls with wall modifiers as tested by us.

To sum up, the practical implications listed above illustrate that reaching semantic as well as pragmatic consensus is a challenge. For better understanding of the issues involved as well as their interdependence and for motivating our approach as well, we present a practical example of the modeling of a wall in the following section, which also accompanies us as a running example throughout the paper. In Section 2.2.1 we will once again return to these issues and formulate three separate technical challenges, for which we will present our solution in the subsequent sections.

1.2. Motivating example

Let us consider the process of exchanging information about a wall between an architect and a structural engineer. Both model the same object, but consider different aspects of it. For the architect (and the building physicists) the wall is a layered construction with thermal, hygric, fire safety and other properties. For the structural engineer the wall is a structural member with a structural behavior within a system. In the IFC standard, there are elements that aid each of them in their modeling task, IfcWallStandardCase and IfcStructuralSurfaceMember, respectively. However, there is no formal mechanism for establishing that both concepts can describe different aspects of the same object, i.e., no possibility for effective integration. This is the semantic heterogeneity. In practice, each domain expert typically works in her own tool and the information exchange is relegated to a BIM Collaboration Format (BCF) file referencing their respective IFC models.

This is a topic also addressed by a data-driven method in the work of [28], which concentrates exclusively on the domains of architecture and structural engineering. The pragmatic heterogeneity occurs due to the assumptions of both stakeholders based on their respective point of view. For example, the structural engineer may view the two semantic concepts as complementary categories of the same object, whereas the architect may regard the one concept as an additional representation of the other. Since these points of view are not formally expressed, the difference is hidden and may give the impression of consensus where there isn’t one.

In summary, this example involves both semantic and pragmatic heterogeneity. We will examine different solutions to the integration challenge in Section 3.3.

1.3. Contribution

Interoperability in BIM requires integration along both the semantic and the pragmatic dimensions. In this work, we present and evaluate a modeling framework capable of working with multiple semantic type systems inhabiting the same syntax in the context of multiple applications. In addition, the framework provides tools for modeling the pragmatic aspects of a data exchange and converting them from an implicit assumption into an explicit formal specification. The combined model of the semantics and pragmatics of a data model provides an integration facade that enables transparency and traceability during interoperability.

The paper is structured as follows. In Section 2, we review the related work and discuss open challenges we face. Section 3 presents our framework step by step by following the workflow for building an integration facade. In this section we describe relevant methods and techniques we employ for the realization of the framework as a prototypical implementation. Section 4 describes the design, evaluation, and results of our conducted case study on the basis of three practical cases. Section 5 concludes this work and outlines future work.

2. Related work

Much of the work on data exchange standards concentrates on bridging legacy, domain or technological gaps. There are several strategies for the implementation of such exchange that have been explored in the past, as shown in the following subsection.

---

22. [https://technical.buildingsmart.org/standards/bcf/](https://technical.buildingsmart.org/standards/bcf/) (last accessed 2021-02-26).
2.1. Data exchange strategies

Here we differentiate between data exchange strategies based on the type of “bridge” offered to cross a certain communication gap. An overview of these strategies is given in Table 2.

2.1.1. Data exchange via common syntax

Data exchange can take place via syntactic containers. Widely used formats employing this strategy include the Co-mma-separated Values (CSV), Extensible Markup Language (XML), the XML Metadata Interchange XML and the Java Script Object Notation (JSON)–see the entry common syntax in Table 2. These formats are often used as storage for arbitrary data, because they offer no semantics of their own. Therefore, there is no interference with the semantics of the stored data. This makes these formats applicable for all domains. Moreover, transformations between them requires establishing only syntactic correspondences, not semantic ones. This transformation task can be automated easily. However, for processing the stored information correctly, each of the interacting applications must have implemented not only the same semantics but, in most cases, also the same serialization routine, e.g., a mandatory ordering of the information.

An example for the strategy of data exchange via common syntax is demonstrated in the Epsilon project [31], in the context of integrating legacy models into new technologies. In the course of this project, Paige et al. [23] implement interoperability between a proprietary modeling tool and an open source model management suite by providing a layer of communication drivers between a Java-based execution engine and a Component Object Model (COM) interface. This layer contains a dedicated driver for each persistence format (e.g., EMF, XML, spreadsheet sheets, etc.). In essence, it demonstrates a technique for loading and manipulating the same semantics by extracting its artifacts from different syntactic containers. Building upon this, the authors use a similar approach in another study, on the integration of Google Spreadsheets coupled with XML-based configuration models. In this study, they bridge the gap between the object-oriented and the table-row-column-oriented (or data-oriented) paradigms via a syntactic translation approach [13]. They do not need to consider the semantics of these formats (since there is none) in order to perform the translation.

2.1.2. Data exchange via common semantics

Implementing not only (i) the same semantics, but also (ii) the same serialization routine, becomes impractical for the extensive semantics typical for the AEC industries. In order to avoid (ii), the data exchange format has to incorporate semantic information in addition to the syntax. An example for this approach is the transfer of geographic data in the domain of urban design. The GeoJSON format builds on the purely syntactical JSON format by adding semantics-carrying keywords to its syntactic structures, e.g., Feature, Point, Polygon, etc. The DXF format, which transfers only geometric information, encodes the geometric semantics in context dependent numerical keys. For example, the key 0 indicates the beginning of an object definition. The key 10 in the context of a line indicates the x-coordinate of its center. The IFC format also belongs to this category (see entry common semantics in Table 2). We already discussed some of its features in Section 1. It has two major advantages over the standards operating on pure syntax: It contains expert knowledge and is maintained by domain experts. This results in a certain level of complexity, which has practical implications. Its textual serialization, specified in ISO 10303-21, contains a keyword for each entity, type, property or quantity set. Thus, for version IFC4 and above there exist over 1500 such keywords. An excerpt of this definition is shown below:

\[
\text{entity_instance = entity_instance_name } \sim \text{ simple_record } \sim \text{ simple_record } = \text{ keyword } \sim \text{ parameter_list } \sim \text{ entity_instance_name } = \text{ 'digit' (digit) }
\]

A major challenge when exchanging data via common semantics is the translation between the common semantics and the internal data model of each application. Even when the data exchange standard is open, this translation remains hidden for the user and this may cause unintended results. For example, if the data exchange standard does not know a specific concept, it cannot transfer it. As an illustration of this drawback let us consider the following scenarios: First, the domain of urban design incorporates concepts for many plants. The IFC standard, however, does not. Therefore, each plant defined in an urban development tool is translated to IFProxy in IFC4, which acts as a placeholder for unknown entities [15]. This loss of information cannot be remedied, even if the data exchange takes place between tools whose internal semantics incorporate the concept of plants. Second, a similar problem occurs when the data exchange standard’s level of detail in the description of a concept differs from that of the internal data model of one of the interacting applications. For example, IFC4 differentiates between the material layers in a wall construction. Tools for calculating thermal flux within a wall construction, however, also differentiate between layers within the same material layer. If the translation from the tool to the standard uses the maximum flux value over all layers within a material layer, it will produce different results from a translation that takes the average flux value over all layers within a material layer.

There are even more complex translation problems, when no clear correspondence between data model elements can be established. For example, if the data exchange standard knows only the triangular mesh as the geometric representation of a surface, a Computer Aided Design (CAD) tool that works with Non-Uniform Rational B-Spline (NURBS) surfaces will have to perform triangulation before passing the surface information to the data exchange standard mesh object. Depending on the parameters and constraints of the triangulation, the resulting meshes of the same surface can differ so much between exports that they cannot be reliably identified as representing the same surface [10].

Ontologies: Operating on semantics brings us to the topic of the application of an ontology in the data exchange. IFC itself is based on an ontology, the buildingSMART Data Dictionary. This approach was chosen by Akamis et al. to address the problem of proprietary data formats resulting in errors and missing data during information exchange for automated cost estimation when employing different BIM software tools (compare to the second limiting factor under entry common semantics in Table 2). In [1], the authors present a novel data-driven method for automated quantity takeoff (QTO), since current methods do not address QTO from BIM artifacts created by different tools. For this purpose, they employ a data-driven reverse engineering algorithm to generate QTO algorithms for any building component by covering the variety of BIM representations via a mapping to IFC. The authors state that their approach “is neither an algorithm nor a software but a method for developing interoperable QTO algorithms”. They argue that their approach is more robust than (traditional) approaches built on proprietary data formats, and therefore, has a higher level of support for interoperability.

In [25], the authors present an ontology-based approach to support change management as well as traceability of changes throughout all design stages. The authors argue that there are so many approaches presenting standardization methods for different file formats and exchanges, but still the document centric approaches result in parsing, interpretation, and serialization problems. Therefore, their approach builds upon

---

4 Leveraging integration façades for model-based tool interoperability

Table 2 Comparison of data exchange approaches.

<table>
<thead>
<tr>
<th>No</th>
<th>Method</th>
<th>Examples</th>
<th>Features</th>
<th>Advantages</th>
<th>Limiting factors</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Common syntax</td>
<td>CSV, XML, JSON</td>
<td>Domain-independent, do not contain a specialized semantics; instead, they offer a structure: a tree, a sequence of containers, etc.</td>
<td>(1) Can store any data and do not interfere with the data semantics. (2) Applicable to all domains. (3) Translation btw. such standards can be easily automated. (4) Contains expert knowledge.</td>
<td>(1) The correct interpretation of the data requires that each tool has a separate implementation of a common semantics or the exactly same serialization routine. (2) Two can become impractical for large domain-specific standards. (3) Depending on the level of detail included in the standard, it can become too large for efficient maintenance. (4) The implementation of the standard involves a translation between the standard and the software’s data model, which can be challenging.</td>
</tr>
<tr>
<td>2</td>
<td>Common semantics</td>
<td>GeoJSON, DIF, IFC, ontology and description logic</td>
<td>Domain-specific, contain (parts of) the domain’s semantics, e.g., a wall, a structural element, etc.</td>
<td>(1) Can store any data and do not interfere with the data semantics. (2) Applicable to all domains. (3) Translation btw. such standards can be easily automated. (4) Contains expert knowledge.</td>
<td>(1) The correct interpretation of the data requires that each tool has a separate implementation of a common semantics or the exactly same serialization routine. (2) Two can become impractical for large domain-specific standards. (3) Depending on the level of detail included in the standard, it can become too large for efficient maintenance. (4) The implementation of the standard involves a translation between the standard and the software’s data model, which can be challenging.</td>
</tr>
<tr>
<td>3</td>
<td>Copy of reality</td>
<td>digital shadow, digital twin</td>
<td>Multiple domain-specific standards are involved, e.g., architecture, geology, building physics, building automation, etc.</td>
<td>(1) Can be maintained and extended by domain experts according to the requirements. (2) Enables automated as well as continuously decision-making. (3) Provides cross-domain traceability. (4) Can be reverse-engineered to give insight into an unfamiliar system.</td>
<td>(1) Does not allow for abstraction and the digital shadow or twin can become exceedingly large (Terabytes or Petabytes in size). (2) The communication between the different domain-specific standards requires a dedicated structure for both hardware and software middleware. (3) Requires a dedicated communications standard. (4) The correct interpretation of the data requires that each tool has a separate implementation of a common semantics or the exactly same serialization routine. (5) Can become very large (Terabytes or Petabytes in size).</td>
</tr>
<tr>
<td>4</td>
<td>Common knowledge</td>
<td>calculation and simulation methods</td>
<td>Can be based on any of the standards listed above.</td>
<td>(1) Contains operational expert knowledge. (2) Can be used for validation and compliance checking. (3) Data model is rarely complex. (4) Can be reverse-engineered to give insight into an unfamiliar system.</td>
<td>(1) Can become very large (Terabytes or Petabytes in size). (2) Can be used for validation and compliance checking. (3) Data model is rarely complex. (4) Can be reverse-engineered to give insight into an unfamiliar system.</td>
</tr>
<tr>
<td>5</td>
<td>Common data</td>
<td>data bus, any domain-independent or domain-specific standard</td>
<td>Can contain any type of monitoring or simulation data, often time-stamped, can be used for automatic semantic classification</td>
<td>(1) Contains operational expert knowledge. (2) Can be used for validation and compliance checking. (3) Data model is rarely complex. (4) Can be reverse-engineered to give insight into an unfamiliar system.</td>
<td>(1) Can become very large (Terabytes or Petabytes in size). (2) Can be used for validation and compliance checking. (3) Data model is rarely complex. (4) Can be reverse-engineered to give insight into an unfamiliar system.</td>
</tr>
</tbody>
</table>

* https://www.w3.org/TR/xmlschema11-1/ (last accessed 2020-11-13).
* https://www.w3.org/TR/rdf-sparql-query/ (last accessed 2020-11-13).
* https://www.w3.org/TR/xmlschema11-1/ (last accessed 2020-11-13).
original (e.g., many details are omitted), and it can represent it under certain circumstances (e.g., during the planning process). However, there are developments in the AEC industries aiming at producing digital twins of buildings [17]. From a BIM perspective, a digital twin is a BIM model with any abstractions, i.e., a copy of reality and this implies that each software that handles the digital twin has to implement the “semantics of everything”.

An example of this development are the numerous attempts to expand the IFC data model by adding more and more detail in various domains [2,9,29]. Many attempts disregarded the International Framework for Dictionaries (IFD) mechanism for defining taxonomies for IFC, and therefore, are inaccessible for other applications (see Section 1.1, (last accessed 2020-11-13)).

The developments described above demonstrate the complexity resulting from overlapping semantic and pragmatic heterogeneity.

2.1.4. Data exchange as knowledge communication

The interest in interoperability in the AEC industries includes not just data but also methods for its manipulation. For example, the calculation of the energy efficiency of a building design involves thermal simulation. IfcBuilding, the so-called ProceduralModel. This model is used to serve as representation of a TunnelElement. Another domain that relies heavily on procedural geometry is urban design. This would require a visible and traceable method for establishing a semantic relationship between the concepts of procedural geometry in underground facilities as well as in urban areas.

Aside from creating a dedicated application, there are multiple approaches to algorithm development in the context of the AEC industries. One of them is the definition of prototypes in applications such as MatLab™ and Modelica. However, such prototypes are generally only accessible via an Application Programming Interface (API) that only few proprietary software integrate. An additional approach is the use of spreadsheets and small code snippets, e.g., in Microsoft Excel™ and Visual Basic for Applications (VBA), respectively. The drawback of this method is the potential for significant unaddressed pragmatic heterogeneity as the link between an ontology node and the model without any abstractions, i.e., a copy of reality and this implies that each software that handles the digital twin has to implement the “semantics of everything”.

An example of this development are the numerous attempts to expand the IFC data model by adding more and more detail in various domains [2,9,29]. Many attempts disregarded the International Framework for Dictionaries (IFD) mechanism for defining taxonomies for IFC, and therefore, are inaccessible for other applications (see Section 1.1, (last accessed 2020-11-13)).

However, such prototypes are generally only accessible via an Application Programming Interface (API) that only few proprietary software integrate. An additional approach is the use of spreadsheets and small code snippets, e.g., in Microsoft Excel™ and Visual Basic for Applications (VBA), respectively. The drawback of this method is the potential for significant unaddressed pragmatic heterogeneity as the link between an ontology node and the model without any abstractions, i.e., a copy of reality and this implies that each software that handles the digital twin has to implement the “semantics of everything”.

An example of this development are the numerous attempts to expand the IFC data model by adding more and more detail in various domains [2,9,29]. Many attempts disregarded the International Framework for Dictionaries (IFD) mechanism for defining taxonomies for IFC, and therefore, are inaccessible for other applications (see Section 1.1, (last accessed 2020-11-13)).

The developments described above demonstrate the complexity resulting from overlapping semantic and pragmatic heterogeneity.

2.1.4. Data exchange as knowledge communication

The interest in interoperability in the AEC industries includes not just data but also methods for its manipulation. For example, the calculation of the energy efficiency of a building design involves thermal simulation. IfcBuilding, the so-called ProceduralModel. This model is used to serve as representation of a TunnelElement. Another domain that relies heavily on procedural geometry is urban design. This would require a visible and traceable method for establishing a semantic relationship between the concepts of procedural geometry in underground facilities as well as in urban areas.

Aside from creating a dedicated application, there are multiple approaches to algorithm development in the context of the AEC industries. One of them is the definition of prototypes in applications such as MatLab™ and Modelica. However, such prototypes are generally only accessible via an Application Programming Interface (API) that only few proprietary software integrate. An additional approach is the use of spreadsheets and small code snippets, e.g., in Microsoft Excel™ and Visual Basic for Applications (VBA), respectively. The drawback of this method is the potential for significant unaddressed pragmatic heterogeneity as the link between an ontology node and the model without any abstractions, i.e., a copy of reality and this implies that each software that handles the digital twin has to implement the “semantics of everything”.

An example of this development are the numerous attempts to expand the IFC data model by adding more and more detail in various domains [2,9,29]. Many attempts disregarded the International Framework for Dictionaries (IFD) mechanism for defining taxonomies for IFC, and therefore, are inaccessible for other applications (see Section 1.1, (last accessed 2020-11-13)).

However, such prototypes are generally only accessible via an Application Programming Interface (API) that only few proprietary software integrate. An additional approach is the use of spreadsheets and small code snippets, e.g., in Microsoft Excel™ and Visual Basic for Applications (VBA), respectively. The drawback of this method is the potential for significant unaddressed pragmatic heterogeneity as the link between an ontology node and the model without any abstractions, i.e., a copy of reality and this implies that each software that handles the digital twin has to implement the “semantics of everything”.

An example of this development are the numerous attempts to expand the IFC data model by adding more and more detail in various domains [2,9,29]. Many attempts disregarded the International Framework for Dictionaries (IFD) mechanism for defining taxonomies for IFC, and therefore, are inaccessible for other applications (see Section 1.1, (last accessed 2020-11-13)).

However, such prototypes are generally only accessible via an Application Programming Interface (API) that only few proprietary software integrate. An additional approach is the use of spreadsheets and small code snippets, e.g., in Microsoft Excel™ and Visual Basic for Applications (VBA), respectively. The drawback of this method is the potential for significant unaddressed pragmatic heterogeneity as the link between an ontology node and the model without any abstractions, i.e., a copy of reality and this implies that each software that handles the digital twin has to implement the “semantics of everything”.

An example of this development are the numerous attempts to expand the IFC data model by adding more and more detail in various domains [2,9,29]. Many attempts disregarded the International Framework for Dictionaries (IFD) mechanism for defining taxonomies for IFC, and therefore, are inaccessible for other applications (see Section 1.1, (last accessed 2020-11-13)).

However, such prototypes are generally only accessible via an Application Programming Interface (API) that only few proprietary software integrate. An additional approach is the use of spreadsheets and small code snippets, e.g., in Microsoft Excel™ and Visual Basic for Applications (VBA), respectively. The drawback of this method is the potential for significant unaddressed pragmatic heterogeneity as the link between an ontology node and the model without any abstractions, i.e., a copy of reality and this implies that each software that handles the digital twin has to implement the “semantics of everything”.

An example of this development are the numerous attempts to expand the IFC data model by adding more and more detail in various domains [2,9,29]. Many attempts disregarded the International Framework for Dictionaries (IFD) mechanism for defining taxonomies for IFC, and therefore, are inaccessible for other applications (see Section 1.1, (last accessed 2020-11-13)).
to produce artifacts conforming to it, e.g., valid instances of the semantic types, containing user input on demand.

**Challenge 2: Exposing the functionality.** In addition to discover the semantics of a software, we need access to its functionality, e.g., the methods that trigger all relevant algorithms. This requires the ability to call said functionality with all necessary data with as little additional effort as possible.

**Challenge 3: Staying up-to-date.** Most standards and software in use undergo constant development for various reasons, often changing their semantics. For example, a change in an algorithm may require a change in the data model. We need to be able to react to such updates as soon as possible. Update cycles of several years, e.g., typical of the IFC standard, place a massive burden on the validity as well as maintenance of already established information flows.

**Challenge 4: Making the pragmatics explicit.** The pragmatics of a standard, concept or software resides in an implicit from in the mind of each user. We need to provide a mechanism for exposing and discussing the pragmatics, in order to reach consensus.

The challenges listed above address a seamless interaction considering the semantics and pragmatics of a single application. Nevertheless, full interoperability requires an unbroken communication network involving multiple applications. This, in turn, requires effective interaction among semantics, which may have partial or full overlap, and between pragmatics that may be contradictory. This brings us to additional challenges concerning (i) translation between semantics, (ii) translation operations in cases where there is no one-to-one correspondence between semantic concepts, and (iii) automation of the translation process. These challenges we will address in our future work.

### 3. Approach

Our approach hinges on the ability to model and manipulate semantic information in a manner that in no way interferes with it. This means that the syntactic container holding such information does not include any additional semantics, but instead, allows depicting structure in an abstract way, similar to XML or JSON. In addition, unlike XML or JSON, the container is modifiable in real-time. This makes model-driven engineering a fitting technique for realizing our approach.

#### 3.1. Model-driven engineering: preliminaries

Based on Martin Fowler’s classification of models as used are as follows:

1. **sketches** for communication purposes, where only partial views of an artifact are specified;
2. **blueprints** to provide a complete and detailed specification of an artifact;
3. **programs** instead of code for the development of an artifact.

This means that during development, a team uses models in several different ways as abstraction of reality, which makes the design of an artifact (e.g., software) a model-driven process. Therefore, models are crucial for understanding and sharing knowledge about a domain of interest. **Model-driven Engineering (MDE)** transforms models into so-called “first-class citizens” in the field of software engineering [1]. The purpose of MDE in the software engineering domain ranges from communication between different stakeholders to the executability of the developed software.

According to [1], there are two main concepts in MDE: models and transformations. The latter is used for employing manipulation operations upon models. The notation for expressing both concepts is known as modeling language. There are different layers of abstraction in MDE. From a bottom-up perspective there are: (i) MO, which contains run-time instances of the defined model elements of the next higher layer M1; (ii) M1, which describes the domain of interest by a domain model and defines the language describing the semantics of that domain; (iii) M2, which defines a modeling language (e.g., UML, SysML, or a domain specific language) for specifying domain models in M1; and the layer (iv) M3, which defines a so-called meta-language for specifying a metamodel such as MOF. It is essential to note, that the concept of the metamodel in the MDE domain differs considerably from the homonymous concept in the AEC domains, where it is sometimes used to describe correspondences or mappings (see [1]). In the MDE domain, a model is an instance of some more abstract model, or metamodel. This is the reason why we could define an infinite number of model levels. However, a model has to conform to its metamodel, which means that all its elements can be expressed as instances of the corresponding elements of the metamodel, as shown in Fig. 1 and implemented in our framework presented in Section 3.2.

In a nutshell, a modeling language is a tool that supports engineers in specifying models, be it in graphical or textual representation.

Transformations are used for mapping between models specified at any level. For example, in model-driven software engineering, transformations are used for the automatic transformation of model elements (M1) to corresponding code statements (also M1), which could be executed at a platform and produce run-time instances corresponding to the model run-time instances (M0). Generally, transformations are defined at a model level higher than the level they are applied on [1]. Transformation rules could be defined manually from scratch, or by defining specific mapping rules by means of a model transformation language such as the Atlas Transformation Language (ATL), which is the most widely used rule-based transformation language, both in academia and in industry. ATL contains a mixture of declarative as well as imperative constructs, is uni-directional (i.e., transformation from language A to language B), and transforms read-only input or source models into write-only output or target models [1].

In the context of this work, we employ the described MDE techniques for extracting a representation of the semantics of any of the involved applications, and for producing valid instances of the types necessary for executing a function call within it. Thereby, MDE enables us to make the sources of semantic conflicts between applications explicit, and to provide a reproducible path towards their resolution. In addition, it provides tools for expressing and negotiating pragmatics. In our case study presented in Section 4, we examine the feasibility of a direct information exchange between applications via our MDE-based framework. We forego of...
employing serialization or using shared memory. Instead, we compile a minimal set of preconditions, coupled with an estimation of the required effort, that need to be met by the involved software to ensure that all input- and output-relevant data structures are exposed as public types, and that there is a suitable entry point. Finally, we evaluate the implications of the presented framework for the data exchange between multiple domains, such as building physics and structural engineering.

3.2. The modeling framework for integration façades

Our approach can be demonstrated by a workflow incorporating our central modeling framework that allows us to overcome the challenges we formulated in Section 2. In this section, we give a broad overview of the workflow as well as the framework and take a deeper look at the framework’s modeling language. We will elaborate on other aspects of our approach step by step in the course of evaluating our case study results in Section 4.

3.2.1. The workflow

Fig. 2(b) shows the structure of our modeling framework on the left, and the type structure of an application written in an object-oriented programming language on the right, as an example. The two structures are similar. Both require a language for describing the structure. In the target application, the instances of the language build a type system. In most cases this is the semantics of the application. The language provides the syntactic containers to hold the semantics. For example, the C# programming language allows us to create classes. The class as a syntactic container resides in the C# language (see the box Language in the top right corner of Fig. 2(b)). Class Person, on the other hand, is a syntactic instance of class and carries semantics representing the concept person in the real world. Class Person would occupy box Types in Fig. 2(b). The same relationship between syntactic containers and semantics holds true in the modeling framework. However, in it, the instances of the language build a model. The Component as a syntactic container resides in the language of the modeling framework (see the box Language in the top left corner of Fig. 2(b) and the top element in Fig. 2(a)). Metamodel Person in analogy to class Person, is a syntactic instance of Component and also carries semantics representing the concept person in the real world. Metamodel Person would occupy the box Metamodel in Fig. 2(b).

As we elaborated in Challenge 1: Exposing the semantics, in order to be able to work with the semantics of any application, it needs to be discovered first. Steps 1, 2 and 3 of the workflow depict the discovery process. Fig. 2(b) also shows the required interfaces. In order to access the target application’s type system without the help of the source code, we need a reflection mechanism or a dedicated API. This is what the first step accomplishes - retrieving information about types, e.g., T1 (see steps 1 and 3 in Fig. 2(b)). In the next step, using a syntax mapping interface integrated into the modeling framework, or its API, we choose the appropriate language elements and build a model of the discovered type system, e.g., the metamodel MM1. Finally, in step 3, we establish a link between each type and its corresponding metamodel. However, this is not yet sufficient to overcome Challenge 1.

The run-time data resides not in the type system of the target application, but in its instances. For this reason, the modeling framework enables the construction of a model of each instance, which can receive user input. This is made possible by the syntactic relationship instance-of implemented by the language of the modeling framework (see relationship OType between Typed Component and Component in Fig. 2(a)). The models of the instances of the target application are constructed as syntactic instances of the metamodel corresponding to its type system (see the box Model in Fig. 2(b)). This enables step 4, in which the user edits the model of each instance. Step 5 uses the established association between each type of the target application and a metamodel to produce models of the run-time instances of the target application’s type system. Those models can inject their information into the target application (step 6), interact with instances produced only in the target application (step 7) and supply data for function calls on the target application (step 8), which result in valid output (step 9). This allows us to overcome Challenge 1.

At this point, let us take a deeper look into the language of the modeling framework. The relevant excerpt is depicted in Fig. 2(a). Its core consists of two classes, Component and Parameter. A component can contain an arbitrary number of parameters and other components. It can also reference other components, be an instance of or the representation of another one. In particular, the syntactic relationships our modeling framework can accommodate are the following:

1. **Association** (unidirectional or bidirectional). This relationship allows coupling, or referencing, between model elements without restrictions. See ref. Components and ReferenceOf in Fig. 2(a).
2. **Containment**. This relationship ensures that one element is completely contained in exactly one other element or not contained in any element at all. See Subcomponents and ContainedParameters in Fig. 2(a).
3. **Instance-of**. This relationship allows one element to be declared as the type of another. See relationship OType between the classes Component and Typed Component in Fig. 2(a).
4. **Representation-of**. This relationship allows one element to be declared as the representation of another. See relationship Representation between the classes Component and Typed Component in Fig. 2(a).

This language allows us to build a model corresponding to any semantic type. Component can model reference-type elements (e.g., a class, which is addressed by reference or pointer), Parameter can model value-type elements (e.g., an attribute of type integer, which is addressed by value). Both Component and Parameter as well as each of the above listed syntactic relationships can act as a carrier for arbitrary semantics. In fact, the differentiation between those four kinds of relationships is motivated mainly by convenience. What we truly need is simply the concept of two syntactic elements having a relationship. The specification of its type can be left entirely to the semantics of its content. For example, in this case, composition relationships between classes are expressed as semantic dependencies being a sub-component of the other, i.e., a true part of it. Association and aggregation relationships, on the other hand, are expressed as one component referencing another. Bidirectional associations require each component to reference the other. The relationship modeling the one between a class and its instances is contained in the OType relationship container. Finally, the relationship modeling the one between a real-world concept and its representation in a model is contained in the Representation relationship container. An example of the application of the language of our framework to model class Person can be found on our project website.26

We will examine the modeling of classes and their instances in more detail in Section 4. Now we turn to the methods of a class and the functionality they implement. Overcoming Challenge 2: Exposing the functionality depends entirely on the target application and the methods of its types. Unless we misappropriate the reflection mechanism to call private methods, we are dependent on its public ones. Moreover, we need to be able to call them with objects, or instances, as input parameters, to enable real-time interaction. This is step 8 of Workflow 1. We will demonstrate two approaches to overcoming this challenge in Section 4, when we discuss specific use cases and programming styles.

26 https://cdl.mint.jku.at/artefacts-for-semantic-integra

challenge-for-big-open-bim Sergei, Reineke, and Schiller
The modeling framework offers all tools to meet Challenge 4: Make the pragmatics explicit since we can enrich the model of the type system of the target application by additional model elements and relationships to add the pragmatics to the semantic model and produce a true integration facade for the target application.

Addressing challenges 1 to 4 will allow us to perform a loss- and distortion-free translation considering both semantics and pragmatics, even in cases where there is no one-to-one correspondence between concepts. We will examine the workflows involved in translation between multiple source and target applications and present their evaluation in our future work.

3.3. Integration facades

Our approach, as described in the previous sections, provides integration facades for data models that include semantics and pragmatics. It is, therefore, necessary to differentiate between semantics, syntax and representation. These three dimensions are generally handled within the same data model, as exemplified by IFC (see Section 1). A separation can significantly simplify the integration process. The pragmatics, on the other hand, consist of implicit assumptions without a formal expression. We will take a closer look after separating the three explicit dimensions. In Fig. 3 (a) we do just that: semantics are depicted along the ontological axis, syntax - along the linguistic axis, as it is the formal language of the data model, and the representation involved in object of the real world is depicted along the conceptual axis. The latter one is where we cross over from abstract modeling into the real world. In Fig. 3 (b) we show the typical case when we model a real-world concept, e.g., that of a wall construction. This concept is represented by a model element, e.g., Ontological Wall Construction. This type can be instantiated as an object in the model, in this case Object wc01 to represent one specific object in the real world, e.g., one specific wall construction that we can interact with in the real world.

However, real world objects are often part of the extension of more than one concept [19]. In our case, the same physical object can be regarded simultaneously as a wall construction and as a vertical slab, as shown in Fig. 3(c). In other words, it has multiple aspects, or it can be viewed from different points of view as discussed in our motivating example (see Section 1.2). On the other hand, the typical object-oriented modeling language does not allow the instantiating of multiple concept representations into the same modeling object. Fig. 3 (d) demonstrates that the instance-of relationship is defined only between one class and one object. Therefore, the typical modeling scenario involves modeling each concept by a separate ontological type, instantiating each type into a separate ontological instance and modeling a connection between them by applying various software patterns. In this case, objects wc01 and vs01 can be used as dynamic types in an object-type pattern, either to assume the role of a dynamic type for the other or to provide multiple dynamic typing for a third object.

In either case, such workaround misuses syntactic tools (the syntactic association relationship) to model semantics (the semantic instance-of relationship). Therefore, a change in the syntax in the process of data exchange can inadvertently cause a hidden change in semantics that can be very difficult to trace and needs handling on a case-to-case basis, depending on the syntax used by the data models of interacting tools. For example, if, during data exchange, we transition from a syntax that allows dynamic associations to a syntax that allows only association queries on demand, the semantic information stored in that association will not always be present to restrict the instance behavior as a true static type would. This would completely remove type safety from the translated data model.

The modeling language we use in our approach offers several additional modeling constructs that avoid this type of intermixing of syntax and semantics. Fig. 4(a) shows an attempt to add a semantic connection to object wc01 and vs01 by adding an additional semantic abstraction level along the ontological axis. We declare that Ontological Type Wall Construction is an instance of Ontological Type Layered Design, Ontological Type Vertical Slab - an instance of Ontological Type Structural Design, and that both are specialisations (or sub-types) of the Ontological Type Design (see Fig. 4(b)). This would allow to establish that wc01 and vs01 have a common type and, in that type’s definition, that they represent aspects of the same real-world physical object. Since the traditional object-oriented paradigm allows only one level of ontological instantiating, between a class and an object, we need a different construct to allow
multiple levels of instantiating. Fig. 4(b) shows that we cannot actually instantiate all ontological types we need by employing the object-oriented approach alone. Fig. 4(c) demonstrates the model-driven engineering approach. The Typed Component element allows an arbitrary number of instantiating steps via the relationship Of Type. Thus, Layered Design, Structural Design and Design can all be syntactically constructed as syntactic instances of Typed Component and semantically connected along the ontological axis.

The declaration of additional types we described above is an example of handling the pragmatic dimension of a data model by making one possible implicit assumption explicit, e.g., that Wall Construction and Vertical Slab are aspects of the same concept (see the motivating example in Section 1.2), and integrating it in the model of the semantics, even if it is not expressed in the type system of the target application.

The pragmatics can be expressed along the conceptual axis as well, since our approach allows representational relationships along the conceptual axis. Fig. 4(d) shows a different type of relationship between the ontological types Wall Construction and Vertical Slab, via the Representing relationship declared between different syntactic instances of element Typed Component. Here, the pragmatics of another user can be made explicit by stating that the vertical slab is regarded as a representation of the wall construction (in a particular context). The situations depicted in Fig. 4(c) and (d) can be regarded as conflicting pragmatics made explicit. This, in turn, makes it possible to resolve the conflict as part of the model and to produce a coherent integration facade of the target application.

Finally, our approach allows us to confine the data model syntax purely to its role as a linguistic tool and to model ontology and representation separately from it. This removes the syntax of the data model from the list of potential semantic error sources during translation between data models.

In the next section, we present an embedded case study consisting of three cases, or units of analysis, on the basis of the guidelines of [26]. The case study enables us to evaluate the approach we presented in this section in a real-world environment, and to examine the separated workflow steps in detail.

4. Case study

The prototype of the modeling framework, we introduced in Section 3 is implemented as a Windows Presentation Foundation (WPF) application in C#. Therefore, we focus our attention on software written in C#, or on applications that offer a C# API for direct access (e.g., Microsoft Excel™).
The three axes of integration continued. (a) adding ontological types (b) connecting the ontological types via a common super-type is not possible with the typical class-object syntax (c) realising the additional types via the syntax of our modeling language (d) realising a representational relationship between ontological types along the conceptual axis.

Fig. 4. The three axes of integration continued. (a) adding ontological types (b) connecting the ontological types via a common super-type is not possible with the typical class-object syntax (c) realising the additional types via the syntax of our modeling language (d) realising a representational relationship between ontological types along the conceptual axis.
4.1. Design

We chose small non-web based simulation software, which is typical for simulation tools in the AEC domains. Usually, such tools are prototypically implemented as part of a research project, but seldom advance from this stage into a product one. Furthermore, as we outlined in Section 1.1, maintaining them in the context of very complex and constantly evolving data exchange standards is not cost-effective. According to our own online search, such tools are effectively not available beyond the duration of the project they accompany.

For this reason, we have chosen two simulation applications available on the website Code Project\textsuperscript{39} that closely mimic the typical features of simulation software in the field of building physics. The first one is a gravity simulator, and the second a sound wave propagation simulator (from the acoustics sub-domain of building physics). Both handle input and output via a Graphical User Interface (GUI). The third application is a Microsoft Excel\textsuperscript{™} tool for calculating the temperature, humidity and CO2 concentration in a single space developed by domain experts at TU Wien [22].

4.1.1. Case 1: a windows forms application simulating particle motion under gravity

This software simulates the motion of a swarm of particles, each with an initial mass and velocity, under the influence of gravity. The result is represented by an animation on a two-dimensional canvas and as a table containing mass, positions and velocities. Both in its functionality and in the presentation of the simulation results, it resembles existing tools in the AEC industries, e.g., a tool for the calculation of light distribution. The main difference lies in the input handling. This tool allows the user to set the initial position of each particle per mouse-click, which makes the result dependent on the user’s hand movements. However, simulation tools in the AEC industries aim to deliver reproducible results. For this reason, they require predictable input, most commonly in the form of a human-readable text file. Therefore, a first step in adapting a tool with an interactive user interface would be to replace the imprecise interaction by precise textual instructions in an input file. This requires the tool to implement custom serialization, which is a non-trivial task depending on multiple factors, e.g., the tool’s data model complexity. In any case, a direct communication with the application’s data model is not possible.

4.1.2. Case 2: a windows presentation foundation application simulating the propagation of sound waves

This software simulates the propagation of sound waves generated by user-defined sound emitting point or line sources. The resulting interference pattern is calculated numerically over a discrete grid and is displayed as an animation on a canvas in one, two, or three dimensions. Its functionality is very similar to existing tools in the AEC industries. There is an entire sub-domain in building physics dedicated to sound protection, which employs similar methods for the calculation of the sound-proofing properties of materials and constructions. As in the previous case, the main difference, from the viewpoint of the user, is the input handling. Sound emitters and sound blockers are defined by mouse-click. An exact numerical definition in a text file is not possible, as the tool is not equipped with a custom serializer.

Cases 1 and 2 will demonstrate each step of Workflow 1 in detail. In particular, we will focus on the part of the implementation that addresses Challenge 1: Exposing the semantics and Challenge 2: Exposing the functionality.

4.1.3. Case 3: a Microsoft Excel\textsuperscript{™} application simulating the thermal behavior of a single space

This case takes a deeper look at a Microsoft Excel\textsuperscript{™} application developed as part of a research project in the domain of building physics [22]. It determines the temperature, humidity and CO2 concentration in a single enclosed space over the course of a week during a heat wave. In addition, it takes climate, wall construction, orientation, user behavior, and building services into account.

Since this configuration is far too complex to depict here, we have used a small example of one possible configuration of the input and output as a representation of the results we obtained in this study. The bottom part of Fig. 5 shows an Excel sheet prepared for the calculation of the U-Value\textsuperscript{40} of a wall construction consisting of two layers. The input cells are green, with each row corresponding to a material layer in the wall construction. The output cell is orange. The green and orange arrows indicate the input and output information flow, respectively. Both the source and the target of the information flow is a wall construction object. In the full case configuration, the input is distributed over multiple sheets and multiple cell ranges within each sheet. The simulation routine is written in a global module in VBA and is called by clicking a button. The output is a time series saved in a dedicated output sheet. The application supplying the input BIM model is object-oriented and defines buildings, spaces, walls etc. as objects.

This case demonstrates the steps involved in addressing Challenge 4: Making the pragmatics explicit. In particular, it focuses on the expression of the pragmatics as a representational relationship. This means it shows the mechanisms involved in declaring one semantic concept as representative of another.

4.2. Evaluation focus

In all three cases, we will make adaptations in the source code and evaluate both the adaptation itself and the implementation of the workflow described in Section 3. During the evaluation, we will underline the aspects that directly address the challenges we listed in Section 2.2.1. Since Challenge 2: Exposing the functionality is entirely dependent on the target application, we will answer the following additional questions:

**CQ 1.** What amount of effort, measured in change in program length as defined in [15], is necessary to expose an alternative entry point in an existing software that accepts an object of arbitrary complexity as input?

---

\textsuperscript{39} https://www.codeproject.com/ (last accessed 2020-11-13).

CQ 2. What amount of effort, measured in change in program length (see above), is necessary to adapt an existing software, not conforming to object-oriented programming conventions, e.g., using arrays of primitive types instead of objects, to an object-oriented entry point requirement?

4.3. Case study procedures

4.3.1. Data collection procedures

The subjects of this study are pieces of software. Our selection criteria included (i) full access to the source code, (ii) the software not being familiar to us or, if it is familiar to us, also having a real-world application, (iii) the software either being of the AEC domains or very similar in function and (iv) the software simulating a physical process based on a one-time input and returning a one-time output. We searched for prototypes of simulation tools from the building physics domain online. For the first two cases we chose applications published on the Code Project Website. For the third, TU Wien provided us with access to a simulation tool for the purpose of this study (the Microsoft Excel™ tool, [22]). In all cases, we were able to examine the code fully and test various adaptations. The purpose of the examination was not to review the quality of the code, but to work with unfamiliar and/or real-world examples and be confronted with real-world challenges.

4.3.2. Analysis procedures

For the analysis of the cases we gathered quantitative data: the change in program length, the change in lines of code, and the number of new types created in the target software (see questions CQ 1 and CQ 2). We further evaluated qualitative data. We tested various approaches to creating an alternative entry point in the target application. We also documented the programming changes required for exposing enough of the underlying semantics through the entry point to enable passing all necessary input data and receiving output.

4.3.3. Validity procedures

The validity of the obtained results can be compromised by several factors. Firstly, the programming expertise of the researchers has an influence on the data collection. For this reason we do not measure the time spent in unsuccessful adaptation attempts, but analyze only the most efficient adaptations achieved. Secondly, the applied technologies can impact both the qualitative and the quantitative evaluation. Quantitative evaluation measured in lines of code can vary greatly based on the programming language and the developer’s programming style. Therefore we have added a more robust measure, the program length, which is defined by [15] as the sum of the total number of operators and the total number of operands in the program.

The creation of the entry point depends strongly on technology as well—therefore we only answer the question, if it is possible, not how expensive it is. The difficulty in exposing the underlying semantics via an entry point, on the other hand, depends on the programming style. We have selected cases with very different programming styles to demonstrate the wide range of challenges an adaptation could face.

4.4. Evaluation

In this section we apply the workflow steps we presented in Section 3 and evaluate the results. We start with Workflow 1 (see Fig. 2) and Case 1 (see Section 4.1.1).

4.4.1. Evaluation of case 1: a windows forms application simulating particle motion under gravity

Workflow 1 starts with the discovery of the semantics of the target application. Step 1 is shown in Fig. 6. The involved parts of the modeling framework and of the target application are highlighted in blue and green, respectively, in the overview image in the top right corner of the figure. Each of the highlighted modules is expanded to show the relevant portion of its content. For example, the semantics of the target application (the green block in the overview image labeled Types) can be seen

Fig. 6. Case 1. Realization of step 1 of Workflow 1.
in detail on the right in the main portion of the figure, in the box with a dashed greed border. This semantic data model consists of three types. There is a Simulation, which refers to Gravity Conditions and contains some Particles. The language of the modeling framework we presented in Section 3 is the in the top left corner of the overview image. The relevant details are depicted on the left in the main portion of the figure.

In addition to the data models, Fig. 6 shows the interfaces involved in step 1 of Workflow 1. On the side of the target application, we use the .NET Reflection API to gather information about the target application’s type system via type Type. The modeling framework realizes the transfer to its own language by employing the types TypeNode, MappingObject and its subtypes MappingContainer and MappingParameter. The application of step 1 of Workflow 1 is depicted as the path highlighted in orange that originates at Type and ends in Component, thereby completing the information transfer from the target application to the modeling framework.

The result from this transfer is shown in Fig. 7. In step 2 of Workflow 1, the instances of TypeNode extract the information necessary both for the construction of the corresponding metamodel of each target type and for its instantiation. Once the type structure has been discovered, the modeling framework uses a subtype of Component, Typed Component, to set a direct relationship between a type and its metamodel element (see the highlighted association OfType between Typed Component and Type in Fig. 7). This is a generic procedure, completely independent of the specific types of the target application. The only requirement is that the application has types exposable by some mechanism, e.g., the meta-information extracted by the Reflection API

Fig. 8 depicts step 3 of Workflow 1. We put particular emphasis on the completed metamodel corresponding to the type system of the target application and represented by the blue block labeled Metamodel in the overview image. It consists of elements M1, M2 and M3. M1 models type Simulation, with attribute Name set to ‘M1’ and attribute OfType-Name set to ‘Simulation’. Effectively, M1 has two types, Typed Component and Simulation. Those types, however, play different roles. The linguistic type is indicated by the dashed orange line connecting M1 and the language element Typed Component. It provides structure, or syntax. The ontological type is indicated by the association OfType between M1 and Simulation highlighted in orange. It determines the meaning, or semantics. Therefore, M1 corresponds to, or is a model of, type Simulation.

Type Simulation includes the attributes Name and Size. Those are modeled by the Parameter instances pM1 and pM2, respectively, contained in M1 via the relationship ContainedParameters. Also included in the structure of M1 is the reference to type Gravity Conditions. It is modeled by M2 of linguistic type Typed Component and ontological type Gravity Conditions. It is associated with M1 via the ref. Components relationship. Fig. 8 shows the model M3 of type Particle as well.

In effect, the metamodel is the model of the target application’s type system. It is shown both in Fig. 8 and in Fig. 9. It enables the creation of run-time instances via reflection. The process begins with the instantiation of this metamodel, or type model, into instance models (see the expanded module Model in blue in Fig. 9).

Let us take a closer look at the instance models in Fig. 9 and compare them to the automatically generated target application run-time instances on the right, in module Instances. Model instance p1 Model is of syntactic type M1, which is itself of semantic type Simulation. The syntactic instance-of relationship between M1 Model and M1 relies on the association OfType between Typed Component and Component in the language utility of the modeling framework (see Fig. 2). The instance-of relationship between an object and its type in the the target application, on the other hand, is enforced by the type system of its programming language. All such relationships are depicted as dashed orange lines annotated ‘instance-of’ in Fig. 9.

The structure of the run-time instance of Simulation, s1, includes the slots Name and Size. Those are modeled by the Parameter instances pM Name and pM Size, respectively, contained in s1 Model via the relationship ContainedParameters. Also included in the structure of s1 is the reference to instance gc of type Gravity Conditions. It is modeled by gc Model of syntactic type M2, associated with M1 via the ref. Components relationship. In addition, Fig. 9 shows the models p1 Model and pb Model of the instance pa and pb of type Particle, respectively. Each slot Mass in a run-time instance is modeled by a Parameter instance with the Name ‘Mass’ and the corresponding value.

Just as the target application permits editing of its run-time data
structure, so does the modeling framework. The size of the simulation, the number and mass of the particles as well as the gravitational conditions can all be adjusted by the user. In steps 4 and 5 of **Workflow 1**, the model instances receive the user’s input (see the orange highlight annotated 4 in the bottom left corner of Fig. 9) and convert it to model instances. In step 6, the corresponding run-time instances of the target application are updated via reflection. In effect, the user input is injected into the target application.

An alternative, simplified solution, which dispenses with the separate modeling of the types and run-time instances of the target application, is shown in Fig. 12 in Appendix A. The correspondence in structure between the target application data structure and its model in Fig. 9 is syntactic, or linguistic. The semantic correspondence is guaranteed by the association to the same types - `Simulation`, `Gravity Conditions` and `Particle`. Both the target application’s data structure and the modeling framework model carry the same meaning - a particle simulation named ‘Earth’ with size 10.5 containing two particles with mass 0.25 and 0.5 and referencing some gravitational conditions.

This concludes the demonstration of steps 1 to 6 of **Workflow 1**. Those steps fulfill the requirements of **Challenge 1**: Exposing the semantics. The modeling framework makes the semantics of the target application available in real-time. This is achieved by producing artifacts conforming to it, e.g., valid run-time instances of its semantic types, containing user input, on demand.

**Challenge 2**: Exposing the functionality requires actual adaptations of the target application and involves steps 7 to 9 of **Workflow 1**. So far, we have gained full access to the run-time instances without implementing any semantics by hand. However, calling an application’s functionality depends on the public methods it exposes. In this case, the application uses Windows Forms for the GUI design. This means that there is a standard entry point without input parameters, which calls the standard constructor of the form:

```
1 [ STAThread ]
2 static void Main () {
3 Application . Run ( new Form1 ());
4 }
```

An alternative entry point, which takes an object, possibly representing the entire simulation, as an input parameter, and calls an adapted form constructor could look like this:

```
1 public static void Go ( Type_1 input ) {
2 Application . Run ( new Form1 ( input ));
3 }
```

The programming style of this tool is object-oriented and there is, in fact, a type representing the entire particle simulation. Therefore, the introduction of the alternative entry point is straightforward. The type representing a particle simulation, `Simulation`, simply replaces `Type_1` in the code above. An additional method for triggering the simulation programmatically completes the list of necessary changes in the source code. After those, in steps 6 to 9 of **Workflow 1**, the tool can be passed input and executed from the modeling framework. The quantitative evaluation of the code adaptation is shown in **Table 3**. The additional type serves to organize input and output and implements the above mentioned method.
In conclusion, we present the adapted workflow. (1) Initially, the modeling framework establishes the connection to the particle simulation tool. (2) It creates a default model of a particle simulation containing only one particle with a default mass zero. This can be viewed as a model of the types of the particle simulation. (3) The user edits the instance model to change the properties of the single particle as well as to add more particles and modify the gravitational conditions. Since the modeling framework enables copying, it means that the user can produce an arbitrary number of simulation instances from that one initial model. (4) The user triggers the simulation with any one of those instances via the alternative entry point Go, which accepts a particle simulation instance as input. (5) The particle simulation tool creates both an animation and a table containing the results.

4.4.2. Evaluation of case 2: a windows presentation foundation application simulating the propagation of sound waves

The programming style of this application is data-oriented. It utilizes numerical methods for the calculation of the interference pattern of sound waves and, therefore, uses a grid, implemented as a two-dimensional array, as its main data structure and input source. There are different approaches to handling two-dimensional arrays, or matrices. If the matrix is sparse, the non-zero entries can be represented as objects. In this case, those would be points in two or more dimensions. In this manner, the entire matrix can be represented by a collection of objects. This scenario is very similar to the one in Case 1. However, instead of using already existing types in the target application, such as Simulation, Gravity Conditions and Particle, we can adapt the source code and add a new type Point2D to represent a two-dimensional emitting point source, and a new type Simulation that wraps a list of Point2D instances.

![Diagram of workflow](image)

**Fig. 9.** Case 1. Realization of steps 4, 5 and 6 of Workflow 1.

<table>
<thead>
<tr>
<th>Metric</th>
<th>Original</th>
<th>Adapted</th>
<th>Diff</th>
<th>Increase in %</th>
</tr>
</thead>
<tbody>
<tr>
<td>Program length</td>
<td>7791</td>
<td>8125</td>
<td>334</td>
<td>4.29</td>
</tr>
<tr>
<td>Lines of code</td>
<td>714</td>
<td>755</td>
<td>41</td>
<td>5.74</td>
</tr>
<tr>
<td>Number of types</td>
<td>8</td>
<td>9</td>
<td>1</td>
<td>–</td>
</tr>
</tbody>
</table>

**Table 3**

Case 1. Quantitative evaluation of the code adaptation.
However, we want to address Challenge 1: Exposing the semantics without modifying the target application. In addition, the scenario described above is less suitable for large dense matrices. Such matrices are produced, for example, by simulation or monitoring software in the building physics domain. A matrix of monitoring measurements taken every 15 s over the course of a year has more than 2 million rows and may have hundreds or even thousands of columns.

The language utility of the modeling framework we presented in Section 3 has classes for handling data-oriented approaches. Those are the sub-types of MultiValue, an abstract container for multiple numeric values, shown in Fig. 10. The values can be organized in scalar fields (MultiValueField), graph fields (MultiValueFunction) or tables (MultiValueTable). An instance of MultiValue can be referenced by an instance of Parameter by means of the ValueField relationship. In this way, it allows the parameter access to all the data via a pointer (see MultiValuePointer). Since a parameter can exist only when contained in a component, the data is automatically associated with one or more components, which can take on multiple roles (see Section 2.1.5). On the one hand, the component can act as a simple container, or wrapper, of the data. On the other hand, if the component has a type, i.e., if it is an instance of Typed Component, it can act as a typing mechanism.

Using a component as a wrapper of the two-dimensional array produces a model comparable to Case 1. Since the data-oriented programming style still uses the C# typing utility, the Reflection API allows the discovery of all relevant types. As we will see below, we need to add at least one new type to the application to act as input parameter of the alternative entry point. We can utilize this additional type, Simulation, as a wrapper for the two-dimensional binary array that carries the input information.

Just as in the previous case, meeting Challenge 2: Exposing the functionality requires code adaptations. The application uses WPF for the implementation of the GUI. This means that the standard entry point is located in an automatically generated file:

```csharp
1 [System.STAThreadAttribute ()]
2 [System.Diagnostics.DebuggerNonUserCodeAttribute ()]
3 [System.ComponentModelUserCodeAttribute ("PresentationBuildTasks ", "+4.0.0.0")]
4 public static void Main () {
5 WpfTransmissionLineMatrixCS.App app =
6 new WpfTransmissionLineMatrixCS.App ();
7 app.InitializeComponent();
8 app.Run ();
9 }
```

An alternative entry point definition may involve generating a new application domain with its own domain manager for the WPF application to run in. The type of the object we use for passing the input to the application has to be declared as serializable; however, no custom serialization needs to be implemented.

```csharp
1 public static void Go(Simulation input)
2 {
3 var dm_type = typeof(DomainManager);
4 string codeBase = Assembly.GetExecutingAssembly().CodeBase;
5 UriBuilder url = new UriBuilder(codeBase);
6 string path = Uri.UnescapeDataString(url.Path);
7 string ab_path = Path.GetDirectoryName(path);
8 var setup = new AppDomainSetup
9 {
10 ApplicationBase = ab_path,
```

### Table 4

| Case 2: Quantitative evaluation of the code adaptation. |
|----------------|----------------|----------------|----------------|
| Metric          | Original | Adapted | Diff. | Increase in % |
| Program length  | 8422     | 8783    | 361   | 4.29           |
| Lines of code   | 1085     | 1123    | 48    | 4.44           |
| Number of types | 5        | 10      | 4     | —              |

Fig. 10. Handling large numeric data in the modeling framework.
4.4.3. Evaluation of case 3: a Microsoft Excel application simulating the
thermal behavior of a single space

In this case, we evaluate the ability of the modeling framework to
overcome Challenge 4: Making the pragmatics explicit. As we laid out in
Section 4.1.3, the setup in Case 3 is complex. The source application has
an object-oriented type system for representing an entire building. The
target application, which simulates the thermal behavior of the building,
consists of multiple Excel sheets and VBA macros. As an illustration of
the task, we chose one small excerpt, which translates a wall construc-
tion to an Excel sheet, as shown in Fig. 5 and, in more detail, in Fig. 11.

On the right in Fig. 11 is the model of the wall construction, con-
sisting of two material layers, of which only one (ml1) is shown. On the
left is the Excel sheet that calculates the U-value of wall constructions.
The translation pairs are connected by colored lines. Green stands for
translation to the spreadsheet, orange for translation from the spread-
sheet. For example, the wall construction we itself maps to the cell under
the label wall constr., by supplying only its ID. Parameter ml1, con-
tained in the material layer ml1, maps to the cell under the label depth,
by supplying only its current numeric value, 0.015. Parameter wcP1,
contained in the wall constr. we, on the other hand, is mapped to
from the cell where the U-value is located in the spreadsheet.

This case example presents us with the task of dealing with a collection
of cells in a table, whose meaning is known only to the designer of the
simulation the table realizes. In essence, we need to extract the pragmatics
and make them explicit in the corresponding model of the modeling
framework. As mentioned in Section 2.1.1, Francis et al. were confronted
with a similar challenge in the case study presented in [13]. Their
approach involved the creation of a metamodel—a Spreadsheet containing
multiple Worksheets, which contain Columns, which can reference each
other by means of Reference. This metamodel allows the automatic
detection of dependences between cells and tables. However, the semi-
antics of those connections still needs to be supplied by the user.

Addressing Challenge 4: Making the pragmatics explicit involves
making the an explicit connection between cells or ranges of cells within
a spreadsheet and a semantic type (see Fig. 4(c)) or concept (see Fig. 4
d)). For example, the user can create a model element in the modeling
framework that corresponds to the concept of an U-Value. Subsequently,
the user can define a semantic instance-of relationship between this
element and the models of cells in a particular range that store U-Values.
This would make the implicit assumptions of the simulation designer
explicit and greatly aid any user by removing a significant potential
error source, the misinterpretation of the simulation’s data model.

This concludes the evaluation of our approach. In the following
subsection we will discuss some threats to validity with respect to the
four challenges (see their definitions in Section 2.2.1) we aim to
overcome.

As mentioned above, the class Simulation (see the input parameter
type in the alternative entry point Go above) plays the role of a wrapper
for the information we need to pass. These adaptations result in the
addition of 4 classes to the tool (see Table 4): two for the entry point, one
as a wrapper for the input and one for translation from the data to the
object-oriented paradigm. Both in terms of lines of code and program
length increase, this case is very similar to the previous one.

This completes the adaptation of Workflow 1 to this case. As in the
previous case, Challenges 1 and 2 were overcome. Challenge 3: Staying
up-to-date is met as a direct consequence of the dynamic connection be-
tween the modeling framework and the target application. Any changes in
the type structure of the target application results in an automatic update
of the dynamically created models in the modeling framework.

In conclusion, we present the adapted workflow:

1. Initially, the modeling framework establishes the connection to
   the sound wave propagation simulation tool.

2. The modeling framework creates an empty table referenced by a
   parameter in a component instance of type Simulation. This
   configuration can be viewed as a model of the default instance of
   the sound simulation.

3. The user populates the table by marking the sound emitting
   sources as ones and leaving all other entries zero.

4. As in case 1, the user triggers the simulation by injecting a model
   of a Simulation instance into the alternative entry point, Go.

5. The tool creates the resulting sound interference pattern.

Fig. 11. Case 3. Translating between type Wall Construction and an Excel sheet.

AppDomainManagerAssembly = dm_type.Assembly. 
AppendDomainManagerType = dm_type.FullName 
}
13 AppDomainManagerDomain = 
AppDomain.CreateDomain( "TempDomain", null,
setup );
14 CallBackContext ctx = new CallBackContext ();
15 ctx.SourceContainer = input;
16 CrossAppsDomainDelegate action = ctx.AppEntry;
17 domain.DoCallBack( action );
18 domain =

As in conclusion, we present the adapted workflow.

1. Initially, the modeling framework establishes the connection to
   the sound wave propagation simulation tool.

2. The modeling framework creates an empty table referenced by a
   parameter in a component instance of type Simulation. This
   configuration can be viewed as a model of the default instance of
   the sound simulation.

3. The user populates the table by marking the sound emitting
   sources as ones and leaving all other entries zero.

4. As in case 1, the user triggers the simulation by injecting a model
   of a Simulation instance into the alternative entry point, Go.

5. The tool creates the resulting sound interference pattern.

Fig. 11. Case 3. Translating between type Wall Construction and an Excel sheet.
4. Evaluation of validity

It is of note that in all three cases we work with applications with clearly defined functionality and a data model consisting of less than 15 types.

4.5.1. Construct validity

As we stated in Section 2.2.1, the goal of our approach is to provide full integration or integration facade for any data model, i.e., a semantic and pragmatic consensus, as this is the backbone of interoperability in any data exchange environment, including Big Open BIM. In order to account for the arbitrary component in our evaluation, in Case 1 and Case 2 we chose applications that were not developed by us or known to us prior to the study. In Case 3 we chose an Excel simulation (see [22]) well known to us. The reason for this is threefold. First, the algorithm and its input and output are far more complex than those in Case 1 or Case 2 — this is the complexity of a real-world applications. Second, the Excel environment is typically used as a prototyping tool in multiple AEC domains. And third, the programming paradigm of this tool is data-oriented instead of object-oriented and gives us the opportunity to account for information flow between semantic-carrying and semantic-agnostic applications.

4.5.2. Hidden factors

In spite of being confronted with a data-oriented programming paradigm in Case 2, we still had the ability, due to the chosen programming language, C#, to utilize object-oriented methods. Our results do not apply to software written in languages that do not allow object-oriented access of any kind.

We have not considered any security aspects in this study.

4.5.3. Generalisation

A data exchange standard cannot rely on our approach unless it is generalizable to include much more complex applications.

Challenge 1: Exposing the semantics. Our approach employs automated methods for extracting the data model of any application. Even a data model as extensive as IFC4.3 can be automatically loaded.

Challenge 2: Exposing the functionality. The three cases we examined allow us to formulate the following requirements that need to be fulfilled by a software with no API, no implementations of memory sharing or of data exchange standards, in order to be fully accessible from other applications:

R2.1. The type structure should contain one type with a pre-defined name, e.g., MainObject, providing controlled access to all objects managing input and output.

R2.2. The application should have an entry point that accepts an instance of type MainObject as input.

R2.3. The application should trigger its main functionality automatically on receiving input over the entry point described above.

Challenge 3: Staying up-to-date. Since both Challenge 1 and Challenge 2 are generalizable, staying up-to-date, in terms of data model and functionality, is generalizable as well.

Challenge 4: Making the pragmatics explicit. The modeling framework allows the definition of additional semantic types as well as relationships along the ontological and conceptual axes. These tools allow the extraction of the pragmatics and making it explicit for all users, thus contributing to the consensus of the integration facade.

4.5.4. Reliability

The researchers involved in this case study have expertise in the following fields: architecture, building physics, model driven engineering, multi-level modeling and ontology engineering.

5. Conclusion and future work

In Section 2.2.1 we formulated four challenges that need to be overcome in any data exchange process in order to achieve uninterrupted multi-directional information flow. This also applies to Big Open BIM in the AEC industries. The modeling framework we presented in Section 3 addresses these challenges to varying degrees, as summarized in Section 4.5. It enables the integration of semantics at run-time and of pragmatics at design-time. The result is that implicit assumptions are made explicit. Consequently, they can be discussed and a consensus both in semantic and pragmatic terms can be reached, since, as stated by Fetzer [12], “meaning is at the heart of both semantics and pragmatics”. In our approach we regard syntax, semantics and representation as the three independent axes of the space in which data exchange operates. This independence, or separation of concerns, allows the semantics to be modeled independently of the notation or representation of information in any available tool, which makes our approach universally applicable to data exchange processes. In addition, we can model pragmatics both along the semantic and the representational dimensions, and produce a full integration of both semantics and pragmatics into an integration facade, which is a prerequisite for producing a single source of truth.

Our future work will focus on utilizing the integration facades in the translation between the data models of various tools and standards, even in cases where no one-to-one correspondence between concepts exists. In addition, we will develop algorithms for the detection of semantic patterns as basis for automated translation procedures and of translation rules between semantic models involving calculations. Furthermore, we will examine alternative approaches to defining a semantic model. In this work we demonstrated the power of Model-Driven Engineering (MDE). We intend to perform a comparison between MDE and ontology design in our future work.

CRediT authorship contribution statement


Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Appendix A. Workflow 1: non-level-respecting modeling style

An alternative, simplified solution to the one in Fig. 9 in Section 4.4.1, which dispenses with the separate modeling of the types and run-time instances of the target application, is shown in Fig. 12. The model presented in the bottom left corner contains both the type and instance information and is ready for user input injection. This is more efficient, however, not level-respecting, model according to [19].
References


4 Leveraging integration facades for model-based tool interoperability

Galina Paskalava, M.Sc. from TU Wien, is a researcher at Montanuniversität Leoben at the chair of Subsurface Engineering. Her main research focus is the application of model-driven software development methods to the data exchange in the domain of Architecture, Engineering and Construction. She is a member of the development team for a multi-stakeholder IT-environment SIMULTAN at TU Wien.

Alessandra Mamak-Hauser, is a professor at Montanuniversität Leoben at the chair of Subsurface Engineering. Before she headed Module 3 in the Christian Doppler Laboratory for Model-Integrated Smart Production (CDL-MINT) at JKU Linz. Her research field focuses on digital transformation in tunnel information modeling as well as in modeling of production systems. Her research interests are on information integration, model-driven engineering and model-based data analytics. She headed numerous interdisciplinary national funded projects in the field of digital transformation.

Manuel Wimmer is a full professor leading the Institute of Business Informatics - Software Engineering at the Johannes Kepler University Linz and he is the head of the Christian Doppler Laboratory CDL-MINT. His research interests comprise foundations of model engineering techniques as well as their application in domain such as tool interoperability, legacy modeling tool modernization, model versioning and evolution, and industrial engineering.

Thomas Bednar, M.Sc. and PhD from TU Wien, is a full professor for Building Physics at TU Wien. His main research focus are prediction models for hygrothermal and acoustic performance of buildings. He has written many papers related to energy use, moisture and building acoustics both on calculations and measurements and is now finishing projects on plus-energy high rise office buildings or safeguarding wooden beams after reconstructions of brick buildings from 1900. He is head of the development of multi-stakeholder IT-environment SIMULTAN at TU Wien for developing and understanding the built environment.
5 A View on Model-Driven Vertical Integration: Alignment of Production Facility Models and Business Models

B. Wally, C. Huemer and A. Mazak;
DOI: 10.1109/COASE.2017.8256235
A View on Model-Driven Vertical Integration: Alignment of Production Facility Models and Business Models

Bernhard Wally¹, Christian Huemer¹ and Alexandra Mazak²

Abstract—Smart manufacturing requires deeply integrated IT systems in order to foster flexibility in the setup, rearrangement and use of attached manufacturing systems. In a vertical integration scenario, IT systems of different vendors might be in use and proprietary interfaces need to be developed in order to exchange information from one system to another. In this paper we present a model-driven approach for vertical integration of IT systems. It is based on the application of industry standards for the representation of hierarchy level specific system properties and an alignment of their key concepts in order to provide bridging functions for the transformation between the different systems.

I. INTRODUCTION

Quite a few names have been coined for the concept of deeply integrated, networked manufacturing systems: industrial internet of things, cyber-physical production systems, digital production, smart manufacturing, or Industrie 4.0—just to name a few. They all share the common vision of the automation and individualization of the complete manufacturing process—from product description, over order processing and production to product delivery. To make this vision a reality, different business partners are required that execute specific processes and provide these capabilities as services. Abstractly speaking, two kinds of system integration are required: horizontal integration for the linking of systems on the same hierarchy level and for seamless communication between different parties and vertical integration for the integration within one partner, from the business floor to the shop floor.

Model-driven engineering (MDE) has developed a rich palette of tools and techniques for the description and manipulation of software models. Formalized cross-disciplinary engineering is supported by translating between the different engineering fields through common meta-metamodels and clearly specified sets of operations for model-to-model transformations, model validations, model querying, etc.

In this work, we will showcase the application of MDE techniques in the field of automated production systems (aPS) and their implications up to the business layer. With regards to “digital production”, the externalizing of internal processes through services that can be queried becomes more and more important. Flexible automation systems that can process and production to product delivery. To make this evolution on

II. RELATED WORK

Since this work is cross-disciplinary, we present the related work in several subsections, starting with information about the chosen metamodels and their features, followed by aligning the contributions of this paper with related work on vertical integration, metamodel alignment, and model co-evolution.

A. CAEX and AutomationML

Computer Aided Engineering Exchange (CAEX) is a data format that has been defined in the scope of IEC 62424-2008 and provides structures (i) for information exchange between Piping and Instrumentation Diagram (P&I) tools and Process Control Engineering (PCE) related Computer Aided Engineering (CAE) tools, as well as (ii) for the representation of PCE requests in P&I diagrams [1]. CAEX is based on XML¹ and enables the metamodelling and modeling of e.g., the hierarchical architecture of a plant, including involved machines and controllers and their physical and logical connections.

IEC 62714 is based on CAEX and defines sets of role classes and interface classes with certain restrictions regarding their application [2], [3]. It is more commonly known as Automation Markup Language (AutomationML, AML), which is the term we will use in the remainder of this paper. AML defines an abstract interface class ExternalDataConnector which is used to reference external documents and elements therein. Two use cases of this external data connector have been defined so far in separate whitepapers: (i) COLLADAInterface specifies how external COLLADA² documents are referenced [4] and (ii) PLCopenXMLInterface defines how PLCopen³ XML documents can be referenced from AML documents [6]. These whitepapers provide a rough guideline on the referencing and integration of external data into AML documents and serve as a starting point for the work presented here.

1Bernhard Wally and Christian Huemer are with the Business Informatics Group, Institute of Software Technology and Interactive Systems, TU Wien, 1040 Vienna, Austria [wally, huemer]@big.tuwien.ac.at.
²Alexandra Mazak is affiliated with the Christian Doppler Laboratory MINT at TU Wien funded by the Austrian Federal Ministry of Science, Research and Economy (BMWFW) mazak@big.tuwien.ac.at.
³cf. https://www.khronos.org/collada/
⁴cf. https://www.w3.org/TR/2008/REC-xml-20081126/
³PLCopen is a vendor- and product-independent association active in industrial control (cf. http://www.plcopen.org/). PLCopen XML is a data exchange format for the storage of PLC program information according to IEC 61131-3 [5].
B. ISA-95 and B2MML

ISA-95 is a series of standards that addresses the integration of the enterprise domain with the manufacturing and control domains. It defines a set of object models for the exchanging of information between these domains—it provides a standard terminology and set of concepts for system integration [7]. The relevant part of ISA-95 for this work is part 2, as it is specified in IEC 62264-2:2013 [8].

Part 2 of ISA-95 specifies common objects and attributes, mainly by a set of commented UML2 class diagrams, that can be roughly differentiated between (i) basic resources that depict the static definitions of an enterprise with regards to its production facilities (e.g., personnel, equipment, and material) and (ii) operations management information that resembles operational data (e.g., operations capabilities, schedules, and performance).

An XML serialization of ISA-95 has been defined in [9], the business to manufacturing markup language (B2MML). The current version of B2MML is compliant with the current version of ISA-95 and has been used in [10] to link ISA-95 information into AML models.

C. Resource-Event-Agent

Resource-Event-Agent (REA) was coined in the early 80's, by consolidating the then current ideas of accounting research within a unified framework [11]. In its initial and very condensed form, REA describes three concepts: economic resources, economic events, and economic agents. Following accounting theory, an economic event resembles something that has actually happened and that causes a record in the general ledger, such as paying for raw material, receiving money for selling finished products, or renting offices.

The initial REA model was extended and refined to a more complete business ontology comprising new types of events for production and a planning layer that allows the specification of contracts, schedules, policies, etc. [12], [13], [14], [15]. REA thus resembles a link between vertical integration (supports the modeling of resource, and horizontal integration (supports the modeling of resource/exchange and internal/external business modeling).

The runtime configurability of well-designed REA-based information systems has been successfully demonstrated in [16], by running a configurable retail information system (RIS) [17]. The RIS domain model can be evolved by adding/removing/altering types, objects, and their properties.

D. Type-Object Modeling Pattern

As a modeling pattern, the type-object (or power type) pattern [18], [19], [20] is often used in runtime configurable systems in order to allow the dynamic creation and manipulation of classifiers (the types) and instances thereof (the objects). The type-object pattern is one solution to the three-level case of multi-level software engineering [21], [22]. The type-object pattern is very convenient for a number of use-cases, which is why it has been adopted in all the previously mentioned technologies: in REA, in ISA-95 (there it is called class layer), and in AML (system unit classes and role classes can be used for creating instances in the type layer).

For REA, the type-object implementation has best been described in [23], [15], [17]. In short, the type-object pattern proposes to provide a type class and an object class for the specification of a specific entity type and its instances. E.g., in order to support multiple types of agents and instances of these agents, a class Agent Type and a class Agent instance (e.g., a person called “John Smith” who is employed as a salesman at a company) would be associated with the specific Agent Type instance “Salesman”. That way, a new type of agent (e.g., “Cashier”) could be added at runtime by creating a new instance of Agent Type; then, Agent instances could impersonate this type of agent.

E. Model-Driven Vertical Integration

The importance of vertical integration in production processes is emphasized in [24] and [25], where a domain specific modeling language, derived from business process model and notation3, is introduced.

In [26] some key challenges for software evolution in aPS are collected, including the co-evolving of interdisciplinary engineering models, which is the challenge addressed in this paper. One of the research goals stated in [26] is the development of automatic consistency mechanisms for domain specific systems. With our approach we can contribute to this research goal.

System evolution in the context of aPS and information systems (IS) is investigated in [27]: (i) hardware changes in a pick and place unit require an evolution of the the state chart model as well as changes at the code level and (ii) the migration of IS components to the cloud demands changes in deployment, configuration, etc. Their approach stresses the importance of architecture models (from IS to control systems) and their use in the estimation of change effort estimation and impact analysis. The examples given comprise manual co-evolution of different systems based on changes in the architectural model. In our approach (i) changes in one system should automatically propagate to other systems, where this is possible and (ii) the overall architectural model is not modeled explicitly, but to be inferred from multiple domain models.

Integration of the various models in aPS is studied in [28] by employing a linking metamodel that allows the explicit linking of model elements from different modeling domains in order to track consistency, constraint satisfaction, etc. Their approach could be used in conjunction with the approach presented in this paper by providing explicit mappings between model elements and not relying solely on the metamodel level. It would be worthwhile to investigate an integration of their linking metamodel into our approach.

Co-evolution of production system models and their libraries is examined in [29], where AML models and their

---


AML model library prototypes are checked for different kinds of inconsistencies. Repair operations are executed or warning messages are displayed using the Epsilon Validation Language6 (EVL). The tools and techniques used in their work overlap with what we have used in our approach, however in contrast to their work, our approach targets the comparison and balancing of diverging metamodels with different main focuses.

In [10], an approach for the alignment of ISA-95 and AML has been proposed, where the different metamodel elements are matched against each other, and a technique is presented to reference ISA-95 information from AML documents. Their work can be used as a basis for the formulation of transformation rules between AML and ISA-95.

A high level alignment of a reduced set of REA and ISA-95 has been presented in [30] and [31], showcasing an application scenario where vertical and horizontal integration are brought together to provide an integrated engineering view on internal processes and external dependencies. Parts of their work are used as a basis for the transformation rules presented in this work.

In [32], the usage of ISA-95 as task layer execution script is exploited, and a high level alignment of REA and ISA-95 is presented. The idea is to rely on ISA-95 for representing detailed production information and provide only relevant information to the business layer. In their work, high level elements of ISA-95 are aligned with REA, while lower level elements of ISA-95 are used to describe aspects that are beyond the usual application of REA. For our approach this means that these lower level concepts are usually not transformed between ISA-95 and REA, but might be relevant for the information exchange between ISA-95 and AML.

III. ALIGNMENT OF AML, ISA-95 AND REA

Given the alignments already defined in [10] (AML and ISA-95) and [30], [31], [32] (REA and ISA-95), we want to showcase how specific entities and their properties propagate between the different systems of interest in order to keep them in sync. Fig. 1 depicts a high level view on the various alignments that are involved in the translation between the systems in vertical integration scenarios.

In the remainder, we are using the Epsilon Language7 (EOL) for querying model states, Epsilon Transformation Language8 (ETL) for model-to-model transformations, and EVL for the validation of models.

Given an entity $PT_{ISAM}$ that resembles an ISA-95 Material Definition with the ID attribute set to “Pine Timber”:

- in AML, this would be correspondingly instantiated as $PT_{AML}$ of type SystemUnitClass with a reference to RoleClass “ProductStructure”;
- in REA, this would be resembled as entity $PT_{REA}$ of type Resource Type.

In order to determine whether a given entity exists in one of the other system levels, we can query its state space and look for a corresponding item. In the case of $PT_{ISAM}$, we can look into AML documents and query the registered SystemUnitClasses for an attribute with the name “ID” and a matching value. It could also be that the link between ISA-95 and AML has been explicitly defined by an ExternalDataReference of a SystemUnitClass to a B2MML document or element representing $PT_{ISAM}$.

In the case of REA, the approach would be similar: the REA model would be searched for a Resource Type with a name or “ID” attribute matching “Pine Timber”. The link could also be explicitly defined in either the REA element using an attribute “ISA-95 ID” that would hold the value “Pine Timber” (cf. Lat. 1). Vice versa, the link could also be defined in ISA-95, by adding a Material Definition Property with the ID “REA ID” and the corresponding value.

Fig. 1. Overview of model links defined for the purpose of vertical integration.

\[\text{Var \ needle = "Pine Timber";}\]
\[\text{Var \ supn = "Material Definition";}\]
\[\text{For \ m in Model.all} ;\]
\[\text{If ( m.resourceTypes }\]
\[\text{.select( n | n.superType.name = supn }\]
\[\text{and n.name = needle }\]
of core concepts. Lst. 2 shows an excerpt of the transformation from an ISA-95 model (exemplified with Material Definitions) into an REA model. The showcased rule creates a new Resource Type and sets its name to the ID of an input Material Definition (line 7). Additionally, the parent Resource Type “superType” is set to the Resource type with name “Material Definition” (line 8–9). Finally, the Resource Type is added to the REA model (line 10). The whole rule is executed only if there exists no other Resource Type with that name already in the REA model (realized through a guard in line 5).

In order to determine if two models of different domains are in sync, they can be validated against each other. Lst. 3 checks whether an REA model under test contains all the Material Definitions of a given ISA-95 model, and using Epsilon² tooling, a human domain expert would get the chance to fix occurring issues with the click of his/her mouse. Lines 5–6 check whether a Resource Type with a name corresponding to the ID of the current Material Definition exists—and if not, it displays the error message defined in lines 7–8: the user interface integration of Epsilon enables executing the fix defined in lines 9–22 (cf. Fig. 2). The fix creates a new Resource Type with a name corresponding to the ID of the current Material Definition (lines 14–15), sets its parent to the generic “Material Definition” Resource Type (lines 16–18) and adds it to the underlying REA model (lines 19–20).

Fig. 2. The error messages of the epsilon validation engine provide a clickable “quick fix” facility.

IV. APPLICATION SCENARIO

An explicit example shall clarify the effect of model-driven synchronized production and business systems under the influence of changes in any of the given subsystems. The use case describes the evolution of involved systems and the services they provide: the addition of a new business service offering requires changes in the production facilities. These changes occur at different hierarchy levels, and we show how these changes can be propagated to systems of other hierarchy levels using model-driven engineering techniques.

A. Initial System State

The example is based on a fictitious company “Glulam Ltd.” that has specialized in the production of glued laminated timber (glulam). The core production process consists of pieces of timber as raw material that are fed into a continuous finger jointer that produces an endless so called “lamella” that is cut into pieces of required length. Several pieces of lamella are then laminated (glued together) to form a thicker piece of wood, a “glulam”, that is often used for building construction work. This production process is depicted in Fig. 3, that roughly correlates with how this process would be modeled in ISA-95 using the Process Segment model. For the sake of simplicity, Personnel Segment Specifications have been omitted.

The corresponding model in REA of the “Lamination” Process Segment is depicted in Fig. 4. It maps to a Transformation Duality Type that consists of a Produce Event Type, a Consume Event Type, and a Use Event Type. The rationale behind this mapping is: a Process Segment is not a specific instance of a production run, but resembles the blueprint for specific Transformation Dualities that relate Transformation Events to each other. Here, the “Lamination” Transformation Duality Type comprises:

- incremental Produce Event Type “Produce Glulam” that indicates the production of a certain amount of “Glulam” Resource Types,
5 A View on Model-Driven Vertical Integration: Alignment of Production Facility Models and Business Models

So far, the glulam lamination process consisted of pressing the lamellas for a given amount of time. However, recently a new type of glue has been suggested to meet the requirements of some important customers with very specific needs. This type of glue requires a minimum temperature of 25°C while curing; however, it yields a much stronger glulam. Therefore, a heating device is installed next to the beam press in order to ensure the required temperature.

1) Evolution of the business system: For the business system, this means adding three new Resource Types “Heating Device”, “Warm Curing Glue”, and “Strong Glulam” and deciding on how the new elements should be integrated into the business model. Here, it is decided to make a copy of the “Lamination” Transformation Duality Type called “Minimum Temperature Lamination” and adapt it by adding and changing Stockflow Types (cf. Fig. 5): (i) the glue to be used is now a warm curing glue, (ii) the final product is now a strong glulam, and (iii) a heating device is added to the Use Event Type.

2) Evolution of the ERP-MES transfer model: For the ERP-MES transfer model (expressed in ISA-95), the heating device needs to be integrated into the production process, and this change must also be reflected in the corresponding ISA-95 models, specifically the material model, the equipment model, and the process segment model need to be changed. Based on an alignment of the meta-models of REA and ISA-95, an initial skeleton of the ISA-95 model can be created by transforming the REA model. The new Transformation Duality Type is transformed into a Process Segment, the Resource Types into elements of the material and equipment model, and the Stockflow Types into Material Segment Specifications and Equipment Segment Specifications. Some transformations require manual intervention or a specific naming or structuring convention, because it is not intrinsically clear, whether a Resource Type is mapped into a Material Class/Definition, Equipment Class, or Physical Asset Class. Lst. 4 shows an excerpt of a cross-model validation between REA and ISA-95, with the convention that the IDs of ISA-95 elements correspond to names of REA elements. The validation code checks whether all REA Transformation Duality Types have a Process Segment counterpart in ISA-95 and if not, offers a quick fix for generating a skeleton Process
Segment. It does that by traversing all Event Types and adding all of their Stockflow Types as Equipment, Physical Asset, or Material Segment Specification. In order to keep the example concise, only the traversal of Use Event Types is depicted.

Lst. 4. Excerpt of a cross-model validation between REA and ISA-95, expressed in EVL. A backslash (\) denotes a soft line break required to adapt to the line width.

```eql
context rea!TransformationDualityType
constraint ProcessSegmentExists |
check : isa!ProcessSegmentType.all.exists( ps | ps.iD.value = self.name ) |
message: "No Process Segment found for **" || self.name || "**"
fix |
| title : "Add missing Process Segment"
do |
| var ps = new isa!ProcessSegmentType; |
| ps.iD = new isa!IDType; |
| ps.iD.value = self.name; |
| for( uet in itself.useEventTypes ) |
| for( sft in uet.stockflowTypes ) |
| var ess = new isa!Equipment\ |
| SegmentSpecificationType; |
| ess.equipmentClassID = new isa! |
| EquipmentClassIDType; |
| ess.equipmentClassID.value = sft.resourceType.name; |
| ess.equipmentUse = new isa! |
| EquipmentUseType; |
| ess.equipmentUse.value = uet.name + ";" + |
| sft.resourceType.name; |
| ps.materialSegment = |
| new isa!MaterialSegment\ |
| Specification.add( ess ); |
| .add( ps ); |

/* Code intentionally left out */
isa!ProcessSegmentInformationType |
| .all.first() .processSegment |
| .add( ps ); |
```

3) Evolution of the Plant Topology: The plant topology that is expressed in AML can benefit from the adapted ERP-MES transfer model by following the mapping rules presented in [10]. As a result, SystemUnitClasses for the heating device, strong glulam, warm curing glue, and minimum temperature lamination need to be created. This can again be achieved by a set of transformation rules that transform an ISA-95 model into a corresponding AML model.

V. CRITICAL DISCUSSION

The presented approach provides an initial setup for the transformation between systems of various layers of automated production systems. Some restrictions apply that prohibit a fully automated transformation between the different systems. E.g., the mapping between REA and ISA-95 sports syntactic inter-model heterogeneities including 1:n, I2I, and BreadthDifference, following the classification presented in [33], regarding the mapping between REA Resource Types and ISA-95 Material Class/Definition, Equipment Class, and Physical Asset Class. The 1:n heterogeneity causes a decision that needs to be taken in order to determine what kind of output instance should be created for a given input instance. One way to solve this issue is by defining company specific conventions or by providing generic Resource Types (e.g., named "Equipment Class") that serve as parent types for respective instances.

Another problem that cannot be handled generically are different levels of indirection, such as REA Events, that have no equivalent in ISA-95. While event entities can be skipped when transforming from REA to ISA-95, they cannot be generically created when transforming from ISA-95 to REA: it is not clear, which ISA-95 Segment Specifications should be bundled together in single Event Types as Stockflow Types and Participation Types. Modeling conventions could help in resolving parts of these issues, or additional reasoning steps could be introduced that would provide a meaningful modeling structure for the entities in question.

We have chosen to present the application of our approach on a very specific fictitious company, however, the approach itself and most of its implementation are company and domain agnostic. This can be verified by examining the code listings and asserting that only metamodel classes and features are referenced, except for e.g., the statement in line 18 of Lst. 3, where a company specific modeling convention is exemplified. This is inevitable in some cases in order to e.g., accommodate to a basic set of rules on how company assets are modeled, or to explicitly denote that specific elements of different domains represent in fact the same entity.

We have refrained from presenting details about the transformation between AML and ISA-95 models, as the alignment between these two metamodels has been presented in more detail in [10] already. Corresponding transformation rules can be inferred from the mappings defined there.

The benefit of our approach is that a common understanding of concepts from different domains is accomplished by relating metamodel elements with each other. This approach is agnostic to the kind of business a company is involved in. Specific implementations could provide industry related information in order to better acknowledge peculiarities and conventions.

VI. CONCLUSION AND OUTLOOK

We have presented a model-driven approach for the co-evolution of models residing on different levels with respect to the automation hierarchy, based on a generic alignment of corresponding metamodels. While the given technique does not provide a “single point of intervention” when it comes to changes in the models, it facilitates the creation of stub models and provides means for cross-model validation. The main contribution is thus the model-driven propagation...
of basic model elements and changes of model elements between models of different hierarchy levels. By defining clear rules it might even be possible to overcome some of the main limitations with respect to the automatic propagation of changes. These limitations might be defined in a generic way, others might only be possible in a company’s environment.

For the future, we would like to specify the cross-model validations and transformations more exhaustively in order to provide a rather complete setup for the co-evolution of models situated in different hierarchy levels in production systems, with the goal of vertical integration. Further, we plan the incorporation of communication stacks such as OPC Unified Architecture. Here, we would like to investigate cross-domain model mining based on runtime information captured in structured communication streams. E.g., the business layer could be adapted via ISA-95 based on operations performance information sourced in the production process.


REFERENCES


6 Flexible Production Systems: Automated Generation of Operations Plans Based on ISA-95 and PDDL

DOI: 10.1109/LRA.2019.2929991
Flexible Production Systems: Automated Generation of Operations Plans Based on ISA-95 and PDDL

Bernhard Wally, Jiří Vyskočil, Petr Novák, Christian Huemer, Radek Šindelář, Petr Kadera, Alexandra Mazak, and Manuel Wimmer

Abstract—Model-driven engineering (MDE) provides tools and methods for the manipulation of formal models. In this letter, we leverage MDE for the transformation of production system models into flat files that are understood by general purpose planning tools and that enable the computation of “plans”, i.e., sequences of production steps that are required to reach certain production goals. These plans are then merged back into the production system model, thus enriching the formalized production system knowledge.

Index Terms—AI-based methods, factory automation, intelligent and flexible manufacturing.

I. INTRODUCTION

Manufacturing systems of the future are required to be more and more flexible, regarding both the products they produce and the production systems themselves [1], [2]. According to the principles of smart manufacturing, products and their recipes are not required to be known at design time, product variants may be edited at runtime, and production planning and scheduling are to be invoked on-the-fly, when a new production order appears. As such, the use of automated planning systems seems very natural, however, current commercial production order appears. As such, the use of automated planning systems seems very natural, however, current commercial industrial planning systems are not sufficient [3].

Processing production orders on-the-fly means that a flexible manufacturing line does not need to be in a predefined initial state before starting a new production. Moreover, the manufacturing line can even be already producing other orders, and thus the state of all resources such as shuttles on a transportation system, or locations of material can vary. Moving these shuttles back to an artificial initial state, as it is done in industrial practice currently, would mean time and energy loss that could and should be avoided. Such high degrees of freedom disqualify traditional ways of programming manufacturing lines and strengthen the need for using automated planning systems being able to react on changing initial conditions and targets. Further, a declarative way of programming related to planners and industrial specification languages is essential for fulfilling the challenging demands of smart manufacturing systems.

In this letter, we are presenting a model-driven approach to automatically transform a manufacturing system specification to a production plan via automated planning. To formulate the manufacturing line planning task, a specification of all industrial components and their actions and interactions is needed. In this environment a number of methods, tools and standards are well established:

1) Production systems engineering: specification of industrial components and processes using industry standards or domain specific modeling languages [4], [5];
2) Model-driven engineering (MDE): generic methods for the specification of discrete models, their validation, manipulation and transformation, etc. [6]–[9];
3) Automated reasoning: methods for realizing reasoning tasks, covering various classes of problems with different computational complexities, from NP-complete propositional logic, EXPSPACE-complete classical planning, semi-decidable first-order logic, up to undecidable halting problems [9]–[11].

Our proposed approach covers both the engineering phase of systems as well as their runtime. With respect to the engineering phase, it can be considered as a verification tool for the fulfillment of functional requirements, with respect to the runtime it can make use of automated reasoning in order to find sequences of production steps to reach initially unknown production system states or to produce products that have not been known at design time. The overview of the proposed approach is depicted in Fig. 1.

This contribution is structured as follows: after a discussion of related work in Section II, Section III describes the rules for transforming production system models into planning problems, while Section IV discusses their application on a specific use case. Section V presents concrete problem statements as well as
Model-driven alignment of structural production system information was further presented in [18]–[20], where business models were aligned to MES models. This letter could be an interesting extension to our work, when it comes to the integration of business information. A compatible ERP-like system has been presented in [21].

B. Automated Planning

Automated planning is a branch of artificial intelligence that deals with the issue of finding plans, which are strategies or sequences of actions. Typical application scenarios are, e.g., plans that are executed by autonomous robots [22]. “Classical” planning [23] describes automated planning where a set of assumptions and restrictions have to hold.

The worst case complexity of classical planning is EXPSPACE-complete, for plan existence problems [24]. On one hand, many planning systems allow to relax some of the “classical” properties that can even lead to semi-decidability [24] of plan existence problems. On the other hand, many well-known planning problems are typically much easier (NP-complete or even better) [25].

The Planning Domain Definition Language (PDDL) is a standardized classical planning language that has been used for approximately 20 years at the international planning competitions. Only some of the automated reasoning methods can be utilized for (semi-)automated solutions at industrial scale. Therefore, we focused on PDDL-based classical planning problems in this letter. The latest version of the language is PDDL 3.1 [26] but there exist many variants/extensions that support various features like goal rewards, probabilistic effects, multi-agent planning, temporal planning, etc. An overview of several extensions of PDDL including explanation of techniques in successful solvers is provided in [27].

A study on usage of PDDL for a collection of typical basic industrial problems is presented in [3]. Compared to [3], the approach proposed in this letter (i) utilizes PDDL as an intermediate format rather than a tool for direct modeling by experts, (ii) we use a real system of industrial scale, and (iii) we are focused on classical (i.e., non-temporary) planners due to the efficiency. While a “bridge between automation and AI planning” is described in [3], we are enhancing this concept by incorporating a standardized specification and its translation into automation and AI planning. Various systems for executing production plans have been proposed. Some of them are discussed in [28] that mainly addresses execution of production plans based on PDDL.

### Fig. 1. High-level view on the proposed approach.

Performance data of our approach. Finally, Section VI concludes and provides hints for future research directions.

## II. RELATED WORK

### A. Model-Driven Engineering

Model-driven engineering (MDE) has long been investigated and practiced. In the early 2000s, standardization efforts finally culminated in widely adopted standards such as the Meta Object Facility and the Unified Modeling Language (UML) [6]. Similar to the object-oriented paradigm of the 1980s ("everything is an object") the new paradigm was “everything is a model" [7]. Based on this foundation, MDE tools may impose domain-specific constraints and perform model checking that can detect and prevent many errors early in the life cycle [8]. This is exactly the main reason why we employ MDE techniques in this letter. We aim to formally provide knowledge of a production system and use this knowledge to verify certain properties thereof.

MDE in the context of smart production is, of course, not new. For instance, IEC 62264 (also known as ISA-95) is a series of international standards describing data structures, activities and a communication protocol in the field of manufacturing execution systems (MES) and their interfaces with enterprise resource planning (ERP) systems [4]. Specifically, parts 2 and 4 of of ISA-95 define a set of UML-based metamodels that enable the modeling of MES related information [12], [13]. With AutomationML, a standardized data format was introduced for representing engineering information in the area of process automation and control [5]. An integration layer for ISA-95 and AutomationML [14] has been presented in [15] and [16], enabling AutomationML to act as a container format for encoding ISA-95 information.

Model-driven transformation of transportation system knowledge from the proprietary tool PX5 Configurator has been discussed in [17], where it was converted into AutomationML before being further processed within another proprietary simulation tool. While we are also using the PX5 Configurator in this letter in our case study, we are not implementing a toolchain to achieve integration between two proprietary tools, but we define generic transformation rules between standardized (modeling) languages.

1 cf. https://www.automationml.org/

2 http://www.icaps-conference.org/index.php/Main/Competitions
Flexible Production Systems: Automated Generation of Operations Plans Based on ISA-95 and PDDL

C. Synopsis

While both, model-driven engineering and automated planning, have been applied to industrial engineering, we are not aware of any particular approach which allows for automated planning solely on the model-based domain representations such as provided by ISA-95. Our proposed approach uses PDDL in a fully transparent way, i.e., the input needed by PDDL solvers is fully derived from the model and also the output provided by PDDL solvers is automatically translated back to the model. Thus, design-time and runtime decisions can be performed by domain experts without requiring knowledge about the underlying solver technology.

III. MODEL-DRIVEN ENGINEERING OF FLEXIBLE PRODUCTION SYSTEMS

Based on the structural description of a production plant sequences of actions shall be derived that enable reaching certain production goals. We have tackled this task by leveraging (i) ISA-95 models of production systems as input and output models and (ii) PDDL as the technology for inferring sequences of actions. For this letter, the most important concepts of ISA-95 are the equipment and process segment models. Among other entities, they provides concepts for describing the machinery available in production environments such as robots, transportation systems, etc., as well as structures depicting the production steps that can be performed using the equipment. In Section III-A we describe the general approach; a detailed description follows in Section III-B, while a concrete example is presented in Section V.

A. Approach

Our model-driven approach requires the formulation of meta-models for the involved domain models. Therefore, we have created metamodels (i) for ISA-95, following the specification given in the standards’ documents and (ii) for PDDL 3.1, based on the Backus–Naur form given in [26].

We are taking two input files into account: (i) an ISA-95 model describing the production system (including equipment, material, process segments, and resource connections) and (ii) one or more ISA-95 models that describe the envisioned goal states. We will show in Section IV-A how the production system model can be automatically derived from a proprietary source model (this is an optional pre-processing step). The output is an ISA-95 model that is derived from the initial ISA-95 model, but now includes information about operations definitions. The applied “core” workflow is depicted in Fig. 2, the individual processing steps (circled numbers) are described below, accordingly.

1) Production system → Planning domain: the production system is parsed and relevant information extracted and transformed into PDDL domain concepts.
2) (Production system + Goal descriptions) → Planning problem: the production system is parsed and relevant information extracted and transformed into the initial state of a PDDL problem. For each goal description that is provided, a separate planning problem is created, with the corresponding goal specifications. The initial state of these planning problems is reused from the initially created PDDL problem.
3) PDDL code generation: so far, the planning domain and problems have been described by means of models. In this step, the models are serialized as plain-text PDDL documents that can be read by standard-conforming PDDL solvers.
4) PDDL solving: for each planning problem a planning solution is calculated by a PDDL solver. If no solution could be found for certain problems, this is also recorded. The solutions are created as plain-text files.
5) Planning solutions → Planning solution models: the plain-text files are “reverse-engineered” into formal PDDL models in order to be useable in the subsequent processing steps.
6) PDDL solution models → Operations data: the sequence of actions found by the solver is transformed into operations that are collected in an ISA-95 model.
7) (Production system + Operations data) → Integrated model: the original production system model and the operations data model are merged into a single ISA-95 model containing both the static production system information and the behavioral information of goal-oriented production steps. Since step 6 generates operations data, it might be desired to generate this data at runtime instead of at design time, in order to enable flexible production systems that are able to compute production plans online. Fortunately, our approach can be applied at design time and at runtime.
We have implemented the workflow previously described based on metamodels of ISA-95 and PDDL that have been formalized using Ecore/EMF. However, our approach could have been realized using any capable technology, including, e.g., ontologies. The transformation of the initial production system information into a planning domain model has been realized threefold, as described in the following two sub-sections for generic information and in Section IV-B for domain-specific concepts. Our approach assumes that ISA-95 ProcessSegments are defined in a way that they refer to EquipmentClasses rather than to Equipment, and that the runtime information uses pieces of Equipment rather than EquipmentClasses. This is typically the case.

1) Metamodel Concepts: relevant metamodel concepts of ISA-95 are converted to certain PDDL statements (cf. Fig. 3). (i) relevant metamodel classes (that are used by the ISA-95 model under observation) are implemented as PDDL Types. (ii) ISA-95 associations are converted to PDDL Predicates. (iii) boolean properties are supported by a dedicated Predicate, e.g., EquipmentPropertyTrue for equipment properties. (iv) for the manipulation of these properties, two Actions are defined: SetEquipmentPropertyTrue and SetEquipmentPropertyFalse. These two actions enable explicit setting of boolean equipment properties that are not tagged with the term pddl:implicit in their description attribute. A PDDL encoding of these transformed concepts is given in Lst. 1. Lines 15–16 and 21–22 encode information that takes into account instance data: properties that are tagged as being set implicitly must not be supported by the generic SetEquipmentProperty* actions.

It is important to note, that this is only one way of encoding an ISA-95 model in PDDL. For instance, boolean properties could instead be translated as specific Predicates and not as objects that are related to equipment instances via generic Predicates.

2) Instance Data: apart from preparing the PDDL environment with generic concept directly inferred from the ISA-95 metamodel, also instance data of the ISA-95 model has an impact on the planning domain description and requires proper mapping (cf. Fig. 4). Examples for the PDDL representation of this mapping are given in Lst. 2, the single mapping statements refer to specific lines in this listing (the instance data used refers to the example given in Section V):

(i) for each class instance (e.g., instances of EquipmentClass, EquipmentClassProperty), a Constant is created, using the instance’s id with a suitable prefix as identifier (lines 2–4). (ii) ProcessSegments are implemented as PDDL Actions, using the id as name (line 7). The process’s segment specifications are converted to parameters, as they represent the required resources for the process (line 8). Relevant ISA-95 relations are checked via specific Preconditions, using the Predicates defined in the metamodel mapping (line 11). Segment specification properties are checked for specific tags that need to be implemented in the ISA-95 model in order for the transformation process to behave as expected: if the pddl:pre or the pddl:post tag is detected, a corresponding Precondition or Effect is created, respectively (lines 12–16 and lines 18–22). Finally, the duration-related attributes of the ProcessSegment are interpreted as cost of the Action, uniformly converted to seconds (line 23).
We are applying the mapping defined above in a use case that is derived from a real production system deployed at the Technical University in Prague, the Industry 4.0 Testbed. It is reduced to only the transportation system and purposely leaves out any robots or material. The physical layout of the chosen use case is depicted in Fig. 5.

Section IV-A explains how a proprietary transportation system model is attached to the workflow as an optional preprocessing step, while Section IV-B explains domain specific knowledge that is to be introduced to the core workflow.

**A. PX5 Configurator**

So far, the process of mapping ISA-95 elements to PDDL has been domain-agnostic. For the chosen use case of the evaluation which is situated in the field of automated intra-logistics, we need to add a few extra conversion rules in order to get meaningful results. For this, it is important to understand how the system under observation works. It is an automated transportation system centered around a monorail track that can carry one or more shuttles. These shuttles can move on the rail between so-called “positioning units” (PU), which are mechatronic systems with a well-defined location on the rail that can physically lock shuttles once they are located at one of these PUs.

In order to simplify the development of a corresponding “Production System” ISA-95 model, we have implemented a converter for the proprietary tool “PX5 Configurator for Montratec”; the conversion workflow is depicted in Fig. 6. In short, we are reading the contents of the PX5 project file, and extracting relevant information in terms of a PX5 model (a corresponding metamodel has been reverse-engineered from the underlying proprietary XML document). Then this PX5 model is transformed into an ISA-95 model and enriched with separately modeled process information. The result of this workflow is an ISA-95 model of a production system that can be used as an input model for the core workflow.

**IV. USE CASE – INDUSTRY 4.0 TESTBED**

We are applying the mapping defined above in a use case that is derived from a real production system deployed at the Technical University in Prague, the Industry 4.0 Testbed. It is reduced to only the transportation system and purposely leaves out any robots or material. The physical layout of the chosen use case is depicted in Fig. 5.

Section IV-A explains how a proprietary transportation system model is attached to the workflow as an optional preprocessing step, while Section IV-B explains domain specific knowledge that is to be introduced to the core workflow.
B. Domain-Specific Concepts

The ISA-95 representation of this transportation system is strongly supported by the concept of ResourceRelationship-Networks. Track elements (straight line, curve and switch) are connected to each other by ResourceNetworkConnection instances of type Track-Connection. Positioning units and shuttles are described as “being attached” to a track element. This is realized by ResourceNetworkConnection instances of type Positioning-Unit-Connection and Shuttle-Connection, respectively, that connect these entities to corresponding track elements. Since multiple positioning units or shuttles can be located at one element, the \((x,y,z)\) coordinates of the entities are stored as FromResourceReferenceProperties. This is required for creating correct routing graphs between the PUs, as well as assigning the shuttles to the correct PUs. In the process of converting an ISA-95 model to PDDL, the track- and PU-connections are simplified to a directed graph containing only PU nodes that are connected with each other. Also, the locations of the shuttles are reduced to those of the PUs, i.e., a shuttle is only in a well-known location if it is physically located at a PU. Locations in-between are not important in the context of our planning problem. The additional mapping rules are described below; they are implemented as part of steps 1 and 2 of the core workflow.

A ProcessSegmentMoveShuttle defines a boolean ProcessSegmentParameter with the id movement and value true. This parameter is recognized in the first transformation step of the core workflow, from the ISA-95 model to the PDDL domain model. This process segment also specifies three EquipmentSegmentSpecifications: the shuttle \(S\) to move and two PUs: the source \(P\) and the destination \(D\). Based on this information, three additional Preconditions and two additional Effects are created. For the preconditions, the following statements are added: first, it is checked whether the shuttle is currently located in the source PU (line 10). Third, it is checked whether the shuttle is not already in the destination location (line 11). The last two statements are somewhat redundant in the case that all actions that manipulate the \(ShuttleLocation\) predicate are correctly implemented (a shuttle should never be in two places at the same time). However, this redundancy can be considered a safety net and might support comprehensiveness for human readers. The effects are clearly related: first, the shuttle is set to be no longer in the source PU (line 14) and second, the shuttle is now located in the destination PU (line 15).

V. Evaluation

We are evaluating our approach threefold: (i) we are evaluating the use case previously described, (ii) we are evaluating similar use cases of various sizes in order to discuss scalability aspects and (iii) we are extending these use cases to include manufacturing operations in order to exemplify transferability of the approach.

All performance numbers mentioned in the remainder have been collected on a standard portable computer, equipped with a 2.4 GHz CPU. Fast Downward\(^4\) was chosen as the PDDL solver, configured with “\texttt{astar(ipdb)}”.

A. Use Case “Industry 4.0 Testbed”

Given an initial state as depicted in Fig. 5, with the four shuttles 1, 2, 3, 4 (encoded as 1234), located in PUs 3, 1, 4 and 2. The question that arose during the design time of this layout was, whether it is possible to bring the shuttles into an arbitrary order, with only one spare PU (5). While the answer to this problem might be obvious to experts, frequently, engineers are not able to answer such questions with confidence, especially if the layout is more complex. Thus, industrial systems are often equipped with additional features in an “ad-hoc” way in the hope that this would solve specific production-related problems. However, there are two problems with this approach: (i) it remains unclear whether the problem is really solved and (ii) these additional features are often quite expensive and might represent over-engineering. Therefore, an approach where the envisioned solution can be formally verified is clearly an advantage.

In our use case we want to find an answer to the question, and we would like to know how expensive (in terms of time) each of the reorderings is, given that each shuttle movement takes 10 s. For that, we have created 23 “goal description” ISA-95 models, each representing one of the desired goal states (excluding the goal state that is equivalent to the initial state): 1243, 1324, 1342, 1423, etc.

Lst. 3 depicts an excerpt of the generated problem definition that formulates the reordering of the shuttles from 1234 to 2341. The initial state is represented in lines 1–4, while the goal description is given in lines 8–11. Other initialization statements are left out—they are generated in step 2 of the core workflow. The 23 generated problem definitions are all exact copies of one

\(4\) cf. \url{http://www.fast-downward.org/}

---

1. \texttt{(predicates)}
2. \texttt{... (ShuttleLocation \texttt{in} \texttt{PU} \texttt{= Equipment})}
3. \texttt{... (ShuttleLocation \texttt{is \texttt{PU} \texttt{= Equipment})}
4. \texttt{... (ShuttleLocation \texttt{is \texttt{PU} \texttt{= Equipment})}
5. \texttt{... (ShuttleLocation \texttt{is \texttt{PU} \texttt{= Equipment})}
6. \texttt{... (ShuttleLocation \texttt{is \texttt{PU} \texttt{= Equipment})}
7. \texttt{... (ShuttleLocation \texttt{is \texttt{PU} \texttt{= Equipment})}
8. \texttt{... (ShuttleLocation \texttt{is \texttt{PU} \texttt{= Equipment})}
9. \texttt{... (ShuttleLocation \texttt{is \texttt{PU} \texttt{= Equipment})}
10. \texttt{... (ShuttleLocation \texttt{is \texttt{PU} \texttt{= Equipment})}
11. \texttt{... (ShuttleLocation \texttt{is \texttt{PU} \texttt{= Equipment})}
12. \texttt{... (ShuttleLocation \texttt{is \texttt{PU} \texttt{= Equipment})}
13. \texttt{... (ShuttleLocation \texttt{is \texttt{PU} \texttt{= Equipment})}
14. \texttt{... (ShuttleLocation \texttt{is \texttt{PU} \texttt{= Equipment})}
15. \texttt{... (ShuttleLocation \texttt{is \texttt{PU} \texttt{= Equipment})}

Listing 3. Domain-specific PDDL snippets.
Listing 4. Excerpt of the PDDL init state and the complete goal description that has been generated from one of the ISA-95 goal description models.

```pddl
1: init
2: (other initialization left out)
3: (Shuttle:location E_Shuttle=01 E_PositioningUnit=00)
4: (Shuttle:location E_Shuttle=02 E_PositioningUnit=01)
5: (Shuttle:location E_Shuttle=03 E_PositioningUnit=01)
6: (Shuttle:location E_Shuttle=04 E_PositioningUnit=02)
7: (goal: and)
8: (Shuttle:location E_Shuttle=02 E_PositioningUnit=03)
9: (Shuttle:location E_Shuttle=03 E_PositioningUnit=03)
10: (Shuttle:location E_Shuttle=04 E_PositioningUnit=04)
11: (Shuttle:location E_Shuttle=01 E_PositioningUnit=02)
```

Listing 5. The generated plan for re-ordering the shuttles from 1-2-3-4 to 2-3-4-1. Note that the chosen solver converts all entities to lower case.

```pddl
1: moveShuttle e_shuttle=01
2: e_positioningUnit=03 e_positioningUnit=00
3: moveShuttle e_shuttle=02
4: e_positioningUnit=01 e_positioningUnit=03
5: moveShuttle e_shuttle=03
6: e_positioningUnit=04 e_positioningUnit=01
7: moveShuttle e_shuttle=04
8: e_positioningUnit=02 e_positioningUnit=04
9: moveShuttle e_shuttle=01
10: e_positioningUnit=04 e_positioningUnit=02
11: ; cost = 50 (general cost)
```

It can be seen that it is feasible to compute transportation plans for the given topology and load factor for settings as large as 10 shuttles on 15 PUs. This would account for small to medium sized systems. In the largest case that has been tested, computation took ≈ 50 s, which can be considered very responsive, given that the plan execution length of the corresponding solution amounts to 810 s. What can also be seen is that the length of the computed plans increases linearly, while the computational effort (memory and runtime) grows exponentially. In order to compute solutions for larger systems, it will be necessary to find either a better streamlined encoding of the ISA-95 model in PDDL, or to divide larger problems into smaller sub-problems and solving them independently from each other.

C. Use Case “Transferability”

Production systems usually handle and alter material, which is why we have created an extended version of the use case previously described. This extended version adds material to the setting, namely wooden boards that are mounted to the shuttles. An additional ProcessSegment `DrillBoard` is defined that can be executed by the production system in order to drill a hole into the board. This process segment requires a drilling robot and the shuttle carrying the board needs to be located in a positioning unit that is within the reach of this robot. In our experiments, we have located the drilling robot next to the top right positioning unit, as is depicted in the lower part of Fig. 8. The results of the experiments are depicted using dashed lines. The task for the solver was to find a sequence of actions that would drill a hole in each of the boards.

Most importantly, the results show that it is feasible to convert ISA-95 models that include both inventory movements and manufacturing operations to PDDL and have it successfully solved. The new concepts (Predicates) and entities (Objects) required to formulate the new kind of operation have a significant impact on the solving performance. In fact, we could not compute a solution for the largest experiment (10:15) within our timeout frame of 300 s, which is why the diagram does not show values for this setting.
VI. CONCLUSION

We have presented a conceptual mapping and a workflow for the transformation of ISA-95 models into a PDDL formalism in order to find sequences of production steps to fulfill certain manufacturing goals. We have successfully tested our approach in a use case where a chosen transportation system layout was tested whether it would fulfill certain logistics requirements.

To verify the proposed approach, we have developed a simple software tool that is able to execute the computed plan with real machinery. While the entire workflow has been tested and verified up to the point where real shuttles move in the aforementioned Industry 4.0 Testbed, we have focused on the conceptual model transformation part in this letter.

It is also worth noting that, while this letter has been focused on transportation systems, the generic approach and mapping strategy between ISA-95 and PDDL can be leveraged for other production-related problems as well. Briefly, we have already considered material manipulation (drilling) in one of our evaluation scenarios. Consequently, in a next step we would like to consider product assembly tasks in the production process. This would enable a flexible manufacturing system to create production plans for assembly-based lot-size 1 products automatically. What would be needed for such a scenario, would be the construction plan of the final product, as well as the consideration of machinery capabilities with respect to assembly operations.

The performance data presented in Section V is based on an non-optimized implementation: (i) the PDDL solver could be tweaked by experimenting with the parameters of its search algorithm. (ii) improvements could be achieved by parallelizing the tasks assigned to the PDDL solver. Currently, the 23 problems are solved sequentially in a simple for-loop—of course, the invocation of the solver could be done for several problems in parallel; most easily based on the number of cores the underlying platform provides. (iii) the encoding of the ISA-95 model in PDDL could be streamlined in a way that is more convenient to the solver (i.e., steps 1 and 2 of the core workflow could be improved). This argument has already been teased in Section III-B, where alternative PDDL encodings for specific ISA-95 constructs are mentioned. Future work could also take into consideration more advanced versions of PDDL that would, e.g., enable the specification of durative actions [29], ultimately supporting parallelism of production tasks at the planning level. While such an approach could lead to finding highly efficient production plans, it might be too computationally expensive. Nevertheless, experiments in this direction seem worthwhile.

REFERENCES


7 Thirteen years of SysML: A systematic mapping study

S. Wolny, A. Mazak, C. Carpella, V. Geist and M. Wimmer;
DOI: 10.1007/s10270-019-00735-y
13 years of SysML: a systematic mapping study

Sabine Wolny1 · Alexandra Mazak1 · Christine Carpella2 · Verena Geist3 · Manuel Wimmer1

Received: 7 March 2018 / Revised: 11 February 2019 / Accepted: 9 April 2019 / Published online: 13 May 2019
© The Author(s) 2019

Abstract
The OMG standard Systems Modeling Language (SysML) has been on the market for about thirteen years. This standard is an extended subset of UML providing a graphical modeling language for designing complex systems by considering software as well as hardware parts. Over the period of thirteen years, many publications have covered various aspects of SysML in different research fields. The aim of this paper is to conduct a systematic mapping study about SysML to identify the different categories of papers, (i) to get an overview of existing research topics and groups, (ii) to identify whether there are any publication trends, and (iii) to uncover possible missing links. We followed the guidelines for conducting a systematic mapping study by Petersen et al. (Inf Softw Technol 64:1–18, 2015) to analyze SysML publications from 2005 to 2017. Our analysis revealed the following main findings: (i) there is a growing scientific interest in SysML in the last years particularly in the research field of Software Engineering, (ii) SysML is mostly used in the design or validation phase, rather than in the implementation phase, (iii) the most commonly used diagram types are the SysML-specific requirement diagram, parametric diagram, and block diagram, together with the activity diagram and state machine diagram known from UML, (iv) SysML is a specific UML profile mostly used in systems engineering; however, the language has to be customized to accommodate domain-specific aspects, (v) related to collaborations for SysML research over the world, there are more individual research groups than large international networks. This study provides a solid basis for classifying existing approaches for SysML. Researchers can use our results (i) for identifying open research issues, (ii) for a better understanding of the state of the art, and (iii) as a reference for finding specific approaches about SysML.

Keywords
SysML · Systematic mapping study · Systems engineering

1 Introduction

The Systems Modeling Language (SysML) is a standard from the Object Management Group (OMG) to support the design, the analysis, and verification of complex systems which may include software and hardware components. SysML reuses parts of UML and additionally offers new language elements like value types, quantity kind, as well as the opportunity to describe the functionality of continuous systems [29]. One of the first intentions for SysML was to give systems engineers a modeling language in hand which is not too software oriented [51]. SysML enables to model a wide variety of systems from different perspectives such as behavior, structure, or requirement. The temporarily last version 1.5 was released in May 2017. SysML has been in place for about thirteen years, and various papers capturing different aspects of this standard have been published at different venues by different research communities. Since SysML is used in multi-disciplinary engineering, there are large application fields where the language is used.

To get a better overview of this huge number of contributions as well as to identify the relevance of SysML in scientific communities, we carried out a systematic mapping study by analyzing the abstracts of the different contributions. The study helps to generate knowledge by determining the application fields in which SysML is commonly used, which research groups are involved, etc. These insights help to iden-
To put the aim of this article in a nutshell, we present inputs as well as outputs of the SysML mapping study and show a comprehensive overview of the evolution of SysML over a period of more than 10 years. Additionally, we identify open issues and discuss these issues in the conclusion of this article with regard to SysML 2.0. According to Kitchenham et al. [28], the findings and outlook may support the work of the following stakeholders:

- Research: Scientists just started with research in the field of SysML may use this study as an overview and starting point for their work. Experienced researchers may also use it as reference to save time for in-depth studies and to accelerate the search for open issues.
- Industry: For industry, the findings give a good outline of the state of the art in SysML research. This may enable to transfer knowledge between academia and industry. Such knowledge transfer may push forward the realization of open issues in the vision of Industry 4.0 and cyber-physical systems [10]. At least, industry stakeholders may identify relevant and suitable research outputs for practical settings.

The remainder of this article is structured as follows: Section 2 discusses the related work. In Sect. 3, we present the research method, define the research questions, and describe the process of conducting the mapping study. In Sect. 4, we describe and analyze the extracted data and visualize the results. Section 5 covers possible threats to validity. In Sect. 6, we present the conclusions and an outlook to future work. In Appendices A and B we present references of all covered SysML papers, a list of books, and theses, which were not part of this survey.

2 Related work

In this section, we give an outline on the method of systematic literature review compared to the method of a systematic mapping study. Furthermore, we take a closer look on these methods applied to UML and to its profiles (e.g., SysML, MARTE).

2.1 Systematic literature review versus systematic mapping study

Evidence-based practices, originating from the medicine discipline, have been widely adopted in software engineering (SE) since 2004. In order to address evidence-based SE in the form of systematic literature reviews (SLRs), the corresponding techniques were re-formulated by Kitchenham [26]. SLR is a well-defined methodology to identify, analyze, and interpret evidences in an unbiased and repeatable way [28]. A large majority of published SLRs in the domain of SE has been performed by following the approaches introduced by Kitchenham et al. [25,27]. In addition, there are some authors who have adopted surveys from medicine [35] as well as from social sciences [46], or they have applied refined guidelines like introduced in [11,14,60].

In this article, we apply a broader form of SLR which is known as systematic mapping study (SMS) according to [8], since our intention is to focus on evidences for a specific research topic instead of answering detailed research questions. Based on a set of primary studies, a SMS identifies gaps in the research area under consideration and discovers potential research trends. By doing so, we follow the guidelines for conducting SMS in SE introduced by Petersen et al. [44,45]. Additionally, we apply the survey of Kuhrmann et al. [31] for performing our SMS for SysML (see Sect. 3).

It seems that there are similarities between SMS and SLR; however, the approaches of these two methodologies and also their goals are quite different. For instance, in contrast to SLR, a SMS uses general research questions to classify and aggregate relevant studies to high-level categories [40].

2.2 SMSs and SLRs applied to UML

In empirical studies concerning the maintenance of UML diagrams and their use in the maintenance of code, Fernández-Sáez et al. [17] conducted a SMS. For this purpose, the authors studied 38 already published studies for discovering an empirical evidence by applying the guidelines of [25]. As a result, the authors identified the need for more experiments and case studies in industrial contexts.

In the particular research field of UML-driven software performance engineering, Garousi et al. [18] conducted a SMS to systematically categorize the current state of the art. Thereby, the authors applied the guidelines provided by Kitchenham and Charters [25] and Petersen et al. [44]. Among others, the authors identified emerging trends in this specialized research field based on a set of 90 (from 114 identified) papers published between 1998 and 2011 [18].

Torre et al. [56] deliver a comprehensive summary of UML consistency rules (regarding the different diagram types) by performing a SMS including 94 primary studies published until December 2012. For their SMS, they used in total seven search engines and followed the guidelines of Kitchenham [25]. There are related research works that address, e.g., a SLR on UML consistency management [33] by covering an earlier publication period (2001–2007), as well as, a SLR about the quality of UML diagrams [37]. Finally mentioned, there exist prior works on empirical evidence related to UML.
in general, e.g., a SLR [9] and a SMS [47], which consider papers on UML properties and features published until 2008.

In the area of Software Product Lines (SPL), a SMS on business process variability is conducted by Valença et al. [57]. This SMS includes 80 primary studies and considers one empirical study on a hierarchical representation method for UML 2.0 activity diagrams. They based their work mainly on the surveys presented in [8,44] as well as on SMS best practice as introduced in [28].

All of these related works have in common that they do not consider SysML as main topic of the survey and that they apply other techniques than we follow in our mapping study. However, they represent interesting related work, not least because UML provides the basis for SysML.

2.3 SMSs and SLRs applied to UML profiles

Ameller et al. [1] classify UML and UML profiles used to specify functional and non-functional requirements based on SMS to assess the state of the art in the development of services-oriented architectures using model-driven development. The authors selected and analyzed 129 papers by adopting the guidelines presented in [25] and those described in [28,44]. There are related SMS investigating the alignment of requirements specification and testing such as presented in [5]. In [52], the authors conducted a survey to examine the use of UML profiles for testing Web services composition.

In the research field of domain-specific languages (DSLs), Nascimento et al. [13] perform a SMS to identify the most popular application domains of DSLs. The authors categorize 1440 (from 4450 identified) primary studies by applying the guidelines described in [25,44]. The technique of UML profiles is mentioned in 21 publications of their catalog. An extensive SLR in the specialized research area of model-driven security was conducted by Nguyen et al. [39], where the authors also consider UML profiles (e.g., UMLSec, SecureUML, etc.) for the definition of security-oriented DSLs. In addition, Souag et al. [55] surveyed UML-based extensions for modeling security in the field of security requirements engineering.

The UML profile SysML is addressed as topic in a mapping study, which investigated the usability requirements elicitation [41]. The study was conducted based on the guideline presented in [25]. The authors formulated a sub-question on notations to elicit usability requirements, and they identified model-based notations and natural language as the most widely used notations in SE. There are similar SLRs related to this topic such as presented in [2], which covers model-driven requirements engineering.

Regarding model-based requirements specifications, Rashid et al. [48] investigated how UML, SysML, and MARTE profiles have been used to specify aspects of embedded systems in the context of early design verification by considering papers published between 2008 and 2015. In an additional SLR on tool selection in model-based systems engineering, Rashid et al. [49] classified selected research work in different categories like “modeling category,” where modeling aspects of embedded systems using UML and its profiles SysML and MARTE were discussed. Additionally to model-based or model-driven requirements engineering and specification, SysML as topic was also investigated in the field of model-based testing like in the work presented in [54]. Wortmann et al. [62] explore in their SMS the state of the art of using modeling languages for model-based systems engineering of smart factories. The authors found out that SysML and its variants play a key role as modeling technique for realizing Industry 4.0 approaches.

In the research field of systems engineering, several SLRs include specific research questions concerning UML as well as its profiles SysML and MARTE. For instance, Guesti et al. [20] conducted a SLR on the topic of describing software architectures for systems of systems (SoS). The authors’ second research question targets the techniques that have been used for describing SoS. They identified that most primary studies use UML or SysML as semi-formal architecture description languages.

In the previous past, SMSs were applied on safety and security topics in the research field of systems engineering. For instance, Nguyen et al. [40] conducted a SMS by covering primary studies that focus on several SysML profiles like SysML-Sec. Other SMSs such as presented in [16,22] only touch SysML in their explanations and findings.

There are a lot of SMSs addressing UML-based approaches and UML profiles (e.g., SysML), e.g., (i) an SMS on functional safety conducted by [7], (ii) a survey on SPL evolution presented by [32], or two SMS on SPL testing conducted by [15,38].

2.4 Synopsis

In this section, we relate existing research to our mapping study. The presented research includes guidelines for conducting SLRs and SMSs such as the work of Kitchenham et al. [25,27] and Petersen et al. [44,45], or Kuhrmann et al [31]. We discussed works including empirical studies, case studies, and surveys on UML and UML profiles, in particular, applied in the domain of software engineering as well as systems engineering. The conducted studies and mentioned surveys investigate in the research fields of requirements engineering, embedded systems in the context of early design verification, model-based systems engineering, security engineering, performance engineering, and software testing, e.g., the quality and usability of UML and UML profiles.

All of these studies and surveys have in common that they do not consider SysML exclusively and that they apply other guidelines than we follow in our mapping study. For
instance, the presented SLRs answer detailed research questions but they give no evidence on various aspects for SysML for systems and software engineering. However, they represent interesting related work and provide relevant entry points to our own mapping study.

### 3 Research method

As research method, we used the previously introduced SMS that enables to cover and classify publications in a specific area. In our study, we are focusing on the abstracts of publications, published in the time period from 2005 to 2017, that have SysML as their main topic.

The process for conducting this SMS is shown as SysML activity diagram in Fig. 1, which mainly bases on the guidelines introduced in Petersen et al. [44]. We modified this mapping process by adapting the last two activities.

Our systematic mapping process consists of five steps (see Fig. 1). It starts by the activity of defining research questions. The output of this activity are appropriate research questions that define the review scope for the next step. In that activity, we conduct a literature search. The output are all publications related to the previously defined research questions. The next step is the screening of those publications in order to select the relevant ones. These relevant publications are the input for the activity called “classification using abstracts,” where we categorize the relevant publications by their abstracts based on the research type facets introduced by Petersen et al. [44] (see Table 1). We enhance this activity by further investigating to classify the abstracts based on systems engineering phases related to the VDI guideline 2206 [58] and contribution types as introduced by Shaw [53]. As output, we get classified abstracts of selected publications, which we use as input for the last activity “mapping of papers.” After this final step, we get a systematic map, which enables us to extract main findings related to our research questions.

In the following, we describe four (Sects. 3.1–3.3) of the five SMS activities based on the research topic of our survey. Afterward, in Sect. 4, we present the final activity “mapping of papers.”

All data (i.e., founded results, search strings, screened paper, classifications) can be also found on figshare at [https://figshare.com/s/871aa0c03aa18eb3edf6](https://figshare.com/s/871aa0c03aa18eb3edf6).

#### 3.1 Activity 1: defining research questions

In this subsection, we define our research questions to specify the review scope of the mapping study and we provide an insight into the intentions behind these questions.

- **RQ 1**: What are the bibliometric key facts of SysML publications?
  The intentions of this research question is to find out (i) the number of SysML publications that were contributed in the period from 2005 to 2017, (ii) the type of those publications (e.g., article, book chapter), (iii) the main venues the publications have been submitted, and (iv) the main research background (i.e., communities) of these venues.

- **RQ 2**: Where are the scientific communities of SysML located and are there main contributors, who scientifically promote SysML topics?
  The intention is to identify and analyze scientific communities working on topics of SysML, e.g., we are interested in the location of these communities. Moreover, we address the question if there are more single authors working on SysML topics, or rather (small) research groups. For instance, we are interested in the number of publications and their authors to identify those publications published by one and the same author. Last but not least, we consider the number of citations of each of the publications to identify the relevance for the respective community. By doing this analysis, we want to find out if there exists a huge network spanning over the world which is working on SysML approaches, or not.

- **RQ 3**: Which research type facets do the identified publications address?
  The main intention is to categorize the different publications by a solid and already well-established schema ([31,59]). Therefore, we use the research type facets introduced by Petersen et al. [44] as described in detail in Sect. 3.3 (see Table 1). Based on this type facets, we want to find out in which research contexts SysML topics are used, e.g., validation, evaluation, etc.

- **RQ 4**: What are the key aspects of applying SysML in the classified publications?
  In addition to assigning the publications to type facets, we are interested to get a deeper insight in the research contribution of those publications. This research question aims to identify (i) in which phase of the engineering process [58] SysML is used, and (ii) the contribution type [53] of the publications.

#### 3.2 Activities 2 and 3: conducting search and screening of publications

After identifying our research questions, the next activity is the definition of appropriate keywords to find all published papers regarding topics about SysML.
Defining Research Questions

Conducting Search

Screening of Papers

Classification using Abstracts

Mapping of Papers

Fig. 1 Activity diagram of the systematic mapping process [44]

Table 1 Research Type Facet [44, p.4]

<table>
<thead>
<tr>
<th>Category</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Validation Research</td>
<td>Techniques investigated are novel and have not yet been implemented in practice. Techniques used are, for example, experiments, i.e., work done in the laboratory</td>
</tr>
<tr>
<td>Evaluation Research</td>
<td>Techniques are implemented in practice, and an evaluation of the technique is conducted. That means, it is shown how the technique is implemented in practice (solution implementation) and what are the consequences of the implementation in terms of benefits and drawbacks (implementation evaluation). This also includes to identify problems in industry</td>
</tr>
<tr>
<td>Solution Proposal</td>
<td>A solution for a problem is proposed; the solution can be either novel or a significant extension of an existing technique. The potential benefits and the applicability of the solution are shown by a small example or a good line of argumentation</td>
</tr>
<tr>
<td>Philosophical Papers</td>
<td>These papers sketch a new way of looking at existing things by structuring the field in form of a taxonomy or conceptual framework</td>
</tr>
<tr>
<td>Opinion Papers</td>
<td>These papers express the personal opinion of somebody whether a certain technique is good or bad, or how things should been done. They do not rely on related work and research methodologies</td>
</tr>
<tr>
<td>Experience Papers</td>
<td>Experience papers explain on what and how something has been done in practice. It has to be the personal experience of the author</td>
</tr>
</tbody>
</table>

Conducting search In contrast to existing work (see Sect. 2), we do not want to cover just a single aspect of SysML. Our aim is to provide an overview of all published papers. Thus, we decide to search for the following keywords:

- SysML
- “Systems Modeling Language” (case insensitive)
- “System Modeling Language” (case insensitive)

There are many different digital literature libraries available on the Web for conducting a literature search. We have opted for the following four established ones:

- Scopus (www.scopus.com)
  One of the largest abstract and citation databases of peer-reviewed literature.

- ACM Digital Library (http://dl.acm.org/)
  ACM is a research, discovery, and networking platform where a collection of full-text articles and bibliographic records can be found.

- IEEE Xplore Digital Library (http://ieeexplore.ieee.org)
  IEEE Xplore provides a full-text access to technical literature in engineering as well as technology.

- DBLP (http://dblp.uni-trier.de/)
  The computer science bibliography database dblp offers open bibliographic information on computer science journals and proceedings.

In our piloting phase, we got more than 2000 papers resulting from the conducted search process in these libraries. In order to obtain more precise results regarding our intention to find out the state of the art of research on SysML in academia, we decided to restrict the search string by the following criteria:

- Publication in the period from 2005 to 2017: The first SysML Specification v.0.9 is online since January 2005. Thus, we use this year as starting point in our systematic mapping study. Since the survey was conducted in late 2017/early 2018, we decided to define 2017 as end date.
- Title: In order to get as output more specific SysML publications, and not only papers mentioning SysML as related

3 http://sysml.org/sysml Specifications/.
work, we restrict the search query to publications where the previously defined keywords are in the title. This decision should ensure that only publications that focus on SysML are included in our result set.

We updated our result set several times to receive a complete set of all relevant publications to answer our research questions. In addition during the revision process, we also made several updates for finding any further publication published in 2017. The final state of our result set, aligned with all libraries, was checked the last time on the 21 of January 2019.

### Screening of publications

For screening the publications, we defined the following exclusion criteria:

- **Duplicates**
  - Papers:
    - without available abstract
    - without an English, German, or French abstract
    - without any context to the language SysML
      For instance, SysML as abbreviation for “System Machine Learning.”
    - with similar abstracts
      Some papers are covering different development stages of a project, and therefore, their abstracts are identical or have been just slightly extended. We deleted the older or shorter version and always kept the newer or longer version in the result set.
  - with identical abstracts
    There are papers with identical content and abstracts; however, they have been published at different venues (e.g., conferences and journals). We decided to leave one of them in the result set and deleted the other publication.
- **Books**: Books are deleted because they are not peer-reviewed (e.g., A Practical Guide to SysML [B4]). The whole list of retrieved books can be found in Appendix B.
- **Theses**: Theses often cover several different aspects and therefore can be assigned to different type facets. In addition, most theses are also (partly) published as conference or journal papers and would be duplicates. Thus, we removed them from the result set. The list of excluded theses can be found in Appendix B.

Based on these exclusion criteria, we double-checked (extractor/checker) all extracted papers in order to ensure that there is consensus on all findings. After performing this screening process, our result set comprises 579 papers. For these papers, we additionally considered the number of citations provided by Google Scholar (see Sect. 4.2). The overall list of the 579 publications is provided in Appendix A.

### 3.3 Activity 4: classification using abstracts

According to the guidelines of Petersen et al. [44], it is sufficient to search only the abstract of a publication for categorization. In order to get a deeper insight of the research context of those publications and for a better mapping, we decide to apply the research type facets of Petersen et al., as presented in Table 1, already in this phase of the SMS. This means that we deviate from the original mapping process by using the research type facets as classification schema to categorize the abstracts of the selected publications. Thus, we modified the activity of “keywording of abstracts” in that we use an already established classification schema.

Besides the assignment of abstracts to these research type facets, it is important to find out in which engineering phase SysML is mainly used and to which contribution type the publications belong in order to get a deeper insight in the research field of the selected publications. For this purpose, we examine the different topics of the papers by analyzing the keywords of the abstracts and cluster the publications based on systems engineering phases and contribution types. In this respect, we adapted the mapping process introduced by Petersen et al. [44] by making a more fine grained categorization, as we describe in the following:

#### Systems engineering phase

Based on the V-model related to the VDI guideline 2206 [58], we distinguish the following phases:

- **Requirements**: Defining the requirements and system properties such as the scope of functions and interfaces.
- **Design**: Designing the architecture of the system.
- **Implementation**: Phase of realization and integration to which simulations and code generators belong.
- **Validation and Verification**: The final phase of the V-model to analyze and check the system.

#### Contribution type

On the basis of the types of research results introduced by Shaw [53], we define our categories for the contribution types. Shaw defines seven different types, in which she is also distinguishing between different data models (empirical, analytic, qualitative model). In our study, we do not focus on different data models. Thus, we adapt these types for our classification process. To give an overview, we briefly describe our contribution types in the following:

3 https://scholar.google.at/.
In this section, we describe analysis and results to answer the research questions (RQ1–RQ4). This output bases on the last activity of our adapted SMS. Additionally, we briefly summarize the main findings related to these questions at the end of each subsection.

4 Mapping of papers

In this section, we describe analysis and results to answer the research questions (RQ1–RQ4). This output bases on the last activity of our adapted SMS. Additionally, we briefly summarize the main findings related to these questions at the end of each subsection.

4.1 RQ 1: bibliometrics of SysML publications

To answer the first research question, we start with analyzing the distribution of published papers in the period from 2005 to 2017. In a second step, we relate the result set to the type of publications. Furthermore, we make a list of venues, where the publications have been submitted. Figure 2 depicts the absolute number of publications per year. The plot shows that this number subjects to fluctuations. We found out that in the years, in which a new version of the SysML standard was published, the number of publications is mostly higher than in the years before. The peak in Fig. 2 indicates that most of the publications were published in 2013.

For further analysis of these results, Fig. 3 illustrates the relationship among the number of publications, the years, and the type of publication. The orange line depicts that in the period from 2005 to 2017 there have been submitted much more inproceedings to scientific conferences than articles to scientific journals (see the blue line) or book chapters (see the green line). Regarding the publication type, 80% of the screened SysML publications were published as inproceeding, 19% as article, and only 1% as book chapter.

Furthermore, we want to find out if there are few selected venues promoted by a handful research communities, or if the publications spread over various conferences, workshops, and journals which are promoted by very different research communities. The result set shows that there are 316 different venues, where the papers have been submitted. For the sake of relevance and clarity, we list in Table 2 those conferences with at least 8 SysML publications and in Table 3 the journals with at least 4 publications. In total, we present 12 different venues. The category column shows the main research community of these venues.

The main venue is the Annual International Symposium of the International Council on Systems Engineering (INCOSE), where more than 30 publications have been submitted in the last 10 years. A possible reason for INCOSE being so prominent may be that the development of the SysML language specification was a collaborative effort between members of OMG and INCOSE. Thus, the INCOSE community is interested in applications and innovations of SysML. Additionally, one of the main research topics of this conference is systems engineering, where SysML plays a key role. From a statistical point of view, we underpin INCOSE’s

<table>
<thead>
<tr>
<th>Venue</th>
<th>Category</th>
<th>Number of Publications</th>
</tr>
</thead>
<tbody>
<tr>
<td>IDETC/CIE (International Design Engineering Technical Conferences)</td>
<td>Sys. Eng.</td>
<td>14</td>
</tr>
<tr>
<td>EITFA (International Conference on Emerging Technologies and Factory Automation)</td>
<td>Aut.</td>
<td>13</td>
</tr>
<tr>
<td>MODELSWARD (Conference on Model-Driven Engineering and Software Development)</td>
<td>Sof. Eng.</td>
<td>11</td>
</tr>
<tr>
<td>CSER (Conference on Systems Engineering Research)</td>
<td>Sys. Eng.</td>
<td>10</td>
</tr>
<tr>
<td>ISSSE (International Symposium on Systems Engineering)</td>
<td>Sys. Eng.</td>
<td>9</td>
</tr>
<tr>
<td>WSC (Winter Simulation Conference)</td>
<td>Sim.</td>
<td>9</td>
</tr>
<tr>
<td>ICEIS (International Conference on Enterprise Information Systems)</td>
<td>Sof. Eng.</td>
<td>8</td>
</tr>
</tbody>
</table>


<table>
<thead>
<tr>
<th>Venue</th>
<th>Category</th>
<th>Number of Publications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Systems Engineering</td>
<td>Sys. Eng.</td>
<td>11</td>
</tr>
<tr>
<td>Innovations in Systems and Software Engineering</td>
<td>Sof. Eng.</td>
<td>5</td>
</tr>
<tr>
<td>Software and Systems Modeling</td>
<td>Sof. Eng.</td>
<td>4</td>
</tr>
</tbody>
</table>

main position by considering the statistical distribution based on the number of papers per venues, listed in Table 2. We get a mean value of 13 (13,1) and a standard deviation of 7 (6,9). Since we identified a spread of 6 to 20 publications per venue in this descriptive statistical analysis, we can classify INCOSE as an outlier compared to the averages of the other conferences.

The second prominent venue is the International Design Engineering Technical Conferences (IDETC/CIE), where 14 papers were submitted and presented. This conference is, among others, one of the main conferences for design engineering mostly related to the manufacturing domain, where SysML fits thematically well, since it is often used in the design phase of automation systems (see Sect. 4.3.1).

The third venue is the International Conference on Emerging Technologies and Factory Automation (ETFA), where 13 papers were submitted. Approaches based on SysML are in line with this conference, since the main topic of this conference is complex systems, and among others, one goal of SysML is to support the modeling of systems considering software as well as hardware components.

All other venues listed in Table 2 have at most 12 publications. Even though these venues are focusing on different subjects, all of them capture the main topics of SysML such as design, simulation, and complex systems. With regard to journals (see Table 3), the listed ones all deal with systems engineering or software engineering topics, whereby the journal Systems Engineering (with a number of 11 publications) has the most published articles with a focus on SysML.

The distribution curve across all these publications (most prominent venues: conferences, journals) in the time frame from 2005 to 2017 regarding the main research communities is shown in Fig. 4. Regarding the number of publications per year, it is obvious that most contributions were published in the field of systems engineering.

Based on the provided information in the relevant abstract of each publication, we classified the publication in various application fields in a double-checking process (extractor/checker). Those publications that do not clearly belong to a specific application field are discussed in the group. If no unambiguous assignment is possible even after this dis-
Figure 4 shows the number of publications per year regarding the research fields of the 12 most prominent venues (included number of studies: 579).

Figure 5 illustrates the number of publications per author (included number of studies: 579).

4.2 RQ 2: research communities and main contributors of SysML topics

In a first step, we analyzed the number of authors and their publications. We identified in total 1167 authors, of whom 30 are single authors without relationship to any other author of the result set. Twenty-seven of these single or “non-related” authors have published only one publication with a SysML topic. The “related” authors have at least one relationship to an author, who also has published a paper about SysML. Figure 6 illustrates the number of publications per author.

It should be noted that most of the related as well as non-related authors (in total 836) have published only one single publication about SysML. However, there are 13 authors, who worked more closely on the topic and wrote at least 10 papers. The specific affiliations of these authors are shown in Fig. 7. It is worth to mention that these 13 authors belong to eight different institutions. Some of them like Hammad,
Mountassir, and Chouali are working together in the same research group, whereas other prominent ones like Paredis, Hause, Vogel-Heuser, and Soares publish on behalf of their own research groups.

For deriving the distribution of related and non-related authors over the world, we took a look at their affiliation to a country. Thereby, we found out that most of the authors are from the USA, followed by France, and Germany (see Fig. 8). Figure 9 illustrates the distribution of authors from a continental perspective. Most of the authors are from Europe (48.8%) followed by North America (23.7%).

In a next step, we analyzed the relationship among these authors to get an overview of networks between them. For this network analysis, we used the free tool Gephi. The results of this analysis are shown in Fig. 10. It illustrates all links among the 1167 authors. A link exists as soon as

---


Fig. 7 Affiliation of authors with at least 10 papers (included number of studies: 579)

Fig. 8 Number of authors per country with at least 5 authors (included number of studies: 579)

Fig. 9 Percentage of numbers of authors per continent (included number of studies: 579)
one author worked with another author on the same publication. Based on Fig. 10, we identified that there are several research networks for SysML, but not a single big one. For a deeper analysis, we took a closer look at the largest network in the entire graph. This research network consists of 61 authors and is shown in Fig. 11. In this figure, we only name the so-called “bridge builders,” who are the authors Paredis from the Georgia Institute of Technology in Atlanta (USA), Friedenthal from the Lockheed Martin Corporation in Fairfax (USA), and Canedo, who is working at the Siemens Corporation Research in Princeton (USA). These three authors are the anchor points linking the research networks across the world.

Finally, we analyzed the influence of the selected publications on the scientific community. We used Google Scholar for counting the citations, since we found no information about citation count in the other used research libraries, except Scopus. In order to make the distribution of citations more comprehensible and to show which publications are cited most frequently in an annual comparison (see Fig. 12, outliers), we have chosen a boxplot for visualization as shown in Fig. 12. The top three papers [12,21,43] are each cited more than 100 times. These three papers are focusing on the SysML topics: simulation, physical systems, and design. These topics, as well as other SysML topics, are one of the most important issues, as the tag cloud shows (see Fig. 14), which we will discuss later on when presenting the results of RQ 4.

**RQ 2—Main Findings:** SysML research takes place worldwide with major contributors especially located in the USA, France, and Germany. However, we discovered that the research interest on SysML topics seems to be stronger in Europe rather than in other continents (see Fig. 9). The social network analysis shows an active community that fostered the discussion on SysML over the years. This network created impact in several engineering domains, ranging from frequently cited basic knowledge (e.g., system and simulation modeling using SysML) to contributions on very specific approaches (e.g., SysML4Modelica, SysML4Mechatronic). In addition, the network analysis shows that there are many smaller research groups working on SysML topics. These groups are partly interconnected by so-called “bridge-builders.” It can be concluded that the interest on SysML has been expanded, since researchers moved to other research groups, bringing their knowledge and interest on SysML topics in these groups in order to work on further SysML approaches.

**4.3 RQ 3: classification of SysML publications**

For the classification process, we used the definition of categories as described in Sect. 3.3. In a first step, each of us
Fig. 13  Number of publications per year according to the type facet classification (included number of studies: 579)

<table>
<thead>
<tr>
<th>Year</th>
<th>Evaluation Research</th>
<th>Validation Research</th>
<th>Solution Proposal</th>
<th>Experience Papers</th>
<th>Philosophical Papers</th>
<th>Opinion Papers</th>
</tr>
</thead>
<tbody>
<tr>
<td>2005</td>
<td>1</td>
<td>4</td>
<td>1</td>
<td>3</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>2006</td>
<td>1</td>
<td>4</td>
<td>7</td>
<td>3</td>
<td>13</td>
<td>4</td>
</tr>
<tr>
<td>2007</td>
<td>27</td>
<td>23</td>
<td>27</td>
<td>10</td>
<td>26</td>
<td>2</td>
</tr>
<tr>
<td>2008</td>
<td>36</td>
<td>30</td>
<td>40</td>
<td>13</td>
<td>22</td>
<td>7</td>
</tr>
<tr>
<td>2009</td>
<td>25</td>
<td>20</td>
<td>25</td>
<td>13</td>
<td>17</td>
<td>9</td>
</tr>
<tr>
<td>2010</td>
<td>30</td>
<td>28</td>
<td>30</td>
<td>6</td>
<td>7</td>
<td>6</td>
</tr>
<tr>
<td>2011</td>
<td>3</td>
<td>5</td>
<td>3</td>
<td>5</td>
<td>9</td>
<td>7</td>
</tr>
<tr>
<td>2012</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>5</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>2013</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>4</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>2014</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>2015</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>5</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>2016</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2017</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

Fig. 14 Tag cloud of most important terms (created with http://tagcrowd.com/, included number of studies: 579)

individually categorized the abstracts of the selected publications based on one of the six different type facets introduced by Petersen et al. [44] (see Table 1). This classification offers the possibility to find out whether SysML is used in own approaches, in experiments, or in theoretical considerations. In a second step, we discussed the categorizations and potential conflicts in the group. Based on these discussions, the conflicting papers were finally assigned to one category. The results of this classification process are shown in Fig. 13.

The result set comprises 10 opinion papers, 32 evaluation papers, 53 philosophical papers, and 60 experience papers. We found out that philosophical papers are so to speak the “pioneers” in the introductory phase of the standard until about 2007. There are negligibly few opinion papers in the result set. The majority of the publications are assigned to the categories solution proposals (185 papers) and validation research (239 papers). This means that the SysML topic of the majority of papers is an own approach and its sample implementation. Figure 13 shows the result set and its chronological sequence which indicates that from 2010 onward validation research and solution proposal become more and more prominent. There are only few evaluation papers, since this category requires a preceding solution implementation and based on this groundwork an evaluation within a practical setting with an industry partner.

RQ 3—Main Findings: The higher number of philosophical papers in the time period from 2005 to 2007 could be explained by the fact that at the beginning of a new modeling language standard the comparison with other modeling standards is usually the focus. Over time when the standard is established, researchers are more interested to take advantage of the standard to realize own approaches and their implementations. This development has been in the foreground since 2010.

4.3.1 RQ 4: key aspects of applying SysML

Additionally to the categorization of publications based on research type facets, we want to detect the key aspects for applying SysML. For this purpose, we firstly created a tag cloud based on all abstracts of the publications of our result set. (German and French abstracts were translated.) Figure 14 shows this tag cloud, which gives us an overview of the 50 most important keywords (with a frequency of at least 50 times) of the abstracts. It should be mentioned that we have deleted conjunctions and keywords like “SysML,” “‘Sys-

![Springer]
conducting the second activity of the mapping process (see Sect. 3.2).

The most frequently used keywords are design, requirements, process, and simulation. Based on the frequency of these keywords, it can be derived that SysML is most frequently used in connection with design and requirement problems. It should be noticed that in this tag cloud every term is counted as often as it occurs in the selected abstracts. In addition, keywords like implementation, verification, and validation frequently appear.

For an even more detailed analysis of the application of SysML topics in the selected publications, we clustered the result set according to systems engineering phases and contribution types, as defined and described in Sect. 3. In the following, we give a summary of the result set analyzed based on engineering phases:

- Requirements: In this initial phase of the engineering process, SysML is used to describe system requirements. The requirements representation is enhanced by a graphical view and by an explicit mapping of the relationships between them. Additionally, the traceability is significantly improved by so-called “requirements tables.” This in SysML newly introduced diagram type helps to bridge the gap between documents written in natural language and modeled use cases. SysML is also used for modeling non-functional requirements. Besides the requirement diagram, the parametric diagram is used to formally describe design requirements for verification and validation purposes.

- Design: We found out that in the design phase, SysML is often used to get a better system understanding and to improve interoperability. In many publications, SysML is used to get a detailed picture of the designed system. Increasingly, SysML is used as modeling language for hardware systems, (ii) for concurrent design processes, (iii) for mechanical concept designs, and (iv) for solving aerospace development problems. Additionally to fulfill special design requirements, SysML is extended by profiles, used in combination with, e.g., MARTE, or mapped to other models.

- Implementation: In the implementation phase, SysML is often used in combination with other languages like SystemC, Modelica, or DEVS to support the implementation of an executable architecture that provides a feasible systems engineering solution. Generally, SysML models are used as basis for the structural and behavioral description of systems. Based on SysML models, executable code is generated by code generators or model transformations are performed by model transformation languages like QVT.

- Validation and Verification: Regarding the V&V phase, we identified that different approaches deal with (i) model checking for the assessment and evaluation of performance characteristics, (ii) generating automated test cases out of models, and (iii) reliability analysis. The formalization of SysML models allows building frameworks for the verification and validation of systems design.

Regarding the research results of the contribution type, there are different research fields addressed in the result set, briefly described as follows:

- Technique: There are a lot of different techniques presented in the publications. Most of them deal with (i) efficient modeling of requirements (functional and non-functional), (ii) performing parametric analysis of complex systems, and (iii) verification of designed models.

- Process: It could be identified that the support of the development process of systems stands in the foreground. Most of the presented approaches deal with the development of requirements up to the entire design phase, whereas only few publications address the process beyond the design phase.

- Notation: We found out that in relation to language engineering, most of the publications of the result set deal with SysML profiles. There are extensions and profiles for (i) facilitating the verification of non-functional quantitative requirements, (ii) improving the application of SysML to complex systems, and (iii) using SysML in the automation, mechatronic system, or embedded system domain. In addition to profiles, there are approaches focusing on translation like transformation to Petri Net or Matlab/Simulink. SysML is also used in combination with OCL, OPM, or MARTE.

- Tool: Approaches in this category mostly engage in the development of tools, e.g., to create and versioning SysML models. There are, for example, requirements modeling tools based on SysML and also tools integrating SysML in a process and design optimization framework. Additionally, there exist approaches that use SysML in combination with simulation frameworks or engines like FUML and James II.

- Specific Solution: There are some specific solutions based on SysML, for example, for space systems, automotive systems, or embedded systems. It can be said that the focus of these solutions is on describing the special requirements of the respective projects.

- Other: The main aspects for assigning publications to the contribution type called “other” are: (i) the comparisons

6 http://jamesii.informatik.uni-rostock.de/jamesii.org/.
of SysML to other modeling languages, (ii) the analysis of the usability of SysML diagrams like requirement view and parametric diagram, and (iii) teaching systems modeling in SysML.

We connected the results of the systems engineering phases and the contribution types together and visualized it in Fig. 15. The distribution shows that most of the publications deal with problems in the design phase followed by the V&V phase. To realize their approaches, the authors mostly develop their own techniques and notations.

There are many papers in our result set dealing with SysML extensions or transformations to other languages, techniques, tools, and concepts. Therefore, we have analyzed the information provided in the abstracts for creating a formalism transformation graph (FTG) [34], and additionally based on the same principle, we created a formalism extension graph (FEG).

The FTG graph in Fig. 16 shows the various transformations of SysML to other languages, techniques, tools, and concepts for different application scenarios such as simulation, verification, analysis, and extracting code. In addition, the FEG in Fig. 17 shows the different extensions of SysML used in the approaches, techniques, and methods introduced and presented in the papers of our result set. Besides the shown transformations and extensions, there are two publications describing linking techniques for SysML to other languages, one to Relax and one to Simulink.

It can be summarized that most of the SysML publications are directed toward individual approaches for the design or validation of systems. In most cases, established languages, mechanisms for extension, and transformations are used. To illustrate these main findings, we give an overall view in Fig. 18 where we show the systematic map of SysML publications regarding type facets, systems engineering phases, and contribution types. This figure presents the interplay of all the probed categories and their classification as output of the last activity of the systematic mapping process (see Sect. 3, Fig. 1).

RQ 4—Main Findings: It turns out that in the area of systems engineering the phases design and validation are predominant topics in all type facet categories. Thus, design and validation are clearly the dominant engineering phases in the usage of SysML. Regarding contribution types, we concluded that the types focusing on technology and notation are more likely to be found in the research type facets “Solution Proposals” and “Validation Research.” In the other type facets categories, the main contribution type can differ. For instance, presentations of specific solutions is an important input for “Experience Papers.”

5 Threads to validity

For identifying the threats of validity of our SMS, we follow the four basic types of validity threats according to Wohlin et al. [61]. We address each of these threats in the following subsections.

5.1 Conclusion validity

Conclusion validity takes care of issues that might arise when drawing conclusions and whether the SMS can be repeated. According to Wohlin et al. [61], the main focus is to draw a correct conclusion regarding relations between the design and outcome of the study. In the given SMS, threats to conclusion validity include:

---

7 Graph also available at https://figshare.com/s/5de5b35ed2ef8ed8f317.
8 Graph also available at https://figshare.com/s/0f0f13ea189b891e312f.
Fig. 16 Formalism transformation graph (FTG) of SysML publications (included number of studies: 579)
Fig. 17 Formalism extension graph (FEG) of SysML publications (included number of studies: 579)

- DEVS – Discrete Event System Specification
- DSL – Domain-Specific Language
- EIS – Enterprise Information System
- MES – Manufacturing Execution System
- SoS – Systems of Systems
- TEPE – Temporal Property Expression
- TLM – Transaction Level Modeling
- OWL – Web Ontology Language
- UPDM – Unified Profile for DoDAF and MODAF
Subjective measures, such as the manual categorization of abstracts to the research type facets of Petersen et al. [44].

Low statistical power, due to the restricted amount of identified publications (e.g., a few publications may influence the ranking of prominent contributors).

Fishing (searching) for specific results, since the results are influenced by the chosen selection of publications (see internal validity).

An additional threat to the validity of the conclusion of a SMS is the publication bias. The term “publication bias” occurs when studies with non-significant findings are either be not submitted by their authors, or may be rejected by reviewers and/or editors and then this could be a risk considering our research type facets. For instance, the risk could be based on the reason that opinion papers are less frequent, since they are more often rejected and become either unofficial technical reports or unpublished studies. To counteract to this risk, we use different databases with various scopes.

As mitigation strategy against subjective measures, the papers of the result set were classified by each of us based on the strategy introduced in Petersen et al. [44], presented in Table 1. Subsequently, these classifications were discussed among themselves. Thereby, occurred discrepancies were considered in more detail and discussed in the group before we re-classified them. Once again, our mitigation strategy against the low statistical power is the use of four different digital libraries to obtain the most complete possible set of papers focusing on SysML as research topic. A comparison with a sampling method such as introduced in [30] would be interesting in order to see whether the same publications would be found. Unfortunately, this investigation goes beyond the scope of this article.

5.2 Internal validity

Threats to internal validity address issues that indicate a causal relationship, such as hidden factors. This phenomenon is also known as spurious correlation. Therefore, the main goal is to guarantee that the methods used in the SMS cause the outcome of the survey. It should be mentioned that factors which impact the internal validity are also significantly influencing the process of the research subjects’ (i.e., publications’) selection. For a better understanding of the internal validity regarding our SMS, we describe in more detail the two influencing factors, selection and instrumentation, based on Wohlin et al. [61], in the following:

Publication selection based on:

- **Keywords**: only the title of publications were searched for the following keywords: ‘SysML’ or ‘System[s] Modeling Language’.

- **Time frame**: restricted from 2005 to 2017, since the first draft of SysML specification was published in 2005. The idea of UML for Systems Engineering was already issued in 2003 but with a different naming.

- **Literature repositories**: we took into account four different literature repositories, which are Scopus, ACM Digital Library, IEEE Xplore Digital Library, as well as DBLP.

- **Publication language**: only publications with English, German, or French abstracts were considered, even though the repositories have provided additional abstracts satisfying the keywords as well as time frame, like Chinese or Spanish publications.

- **Manual filtering**: we deleted duplicates, books, and theses, as well as publications without abstract or research context to SysML.

Instrumentation caused by design of artifacts:

This includes, for instance, **timeliness and completeness** of literature repositories to answer the question which venues are considered by those libraries. It may be possible to delay previously published articles like in the case of post-proceedings, and therefore, they are not available online.

Our mitigation strategy to address risks of publication selection and instrumentation was to avoid too tight restrictions by considering alternatives. For instance, (i) three different keywords based on our mapping scope were used for the search process, (ii) a time frame was applied that started with the first draft of SysML, and (iii) four broad-based literature repositories were taken into account for conducting the search. In contrast to [63], we use the four libraries ACM, IEEE, DBLP, and Scopus and not, for example, SpringerLink. However, SpringerLink references are included in DBLP and Scopus and therefore implicit in our mapping study. In addition, Scopus contains many publications in the field of systems engineering that do not appear in the other libraries. Thus, by this mitigation strategy, the result set may cover a representative set of relevant publications.

Regarding manual filtering, a certain bias remains according to publications with heterogeneous titles and abstracts, but identical content. We discussed this issue in the group. However, this uncertainty remains open due to method we have chosen for this SMS based on Petersen et al. [44].

5.3 Construct validity

Construct validity concerns the relationship between theory and observation. According to Wohlin et al. [61], construct threats to validity cope with issues that might arise during research design. Thus, it should be checked if the used concept is sufficient. There are two kinds of threats to construct
validity, which are (i) design threats such as mono-operation bias, mono-method bias, or confounding constructs as well as levels of constructs, and (ii) social threats such as hypothesis guessing, evaluation apprehension, and experimenter expectancies [61]. It should be mentioned that social threats do not apply to non-personal subjects (such as publications); however, they may be relevant regarding the authors of this mapping study [61]. In the given SMS, threats to construct validity include:

- **Mono-method bias**: the study is mainly based on the systematic mapping process introduced by Petersen et al. [44]. In this context, the mitigation strategy was that two independent research groups have worked cooperatively in this study. In doing so, the first literature review has been independently carried out by each group.

- **Confounding constructs and levels of constructs**: for instance, in the case of categorization, there is more than only one type facet applicable (e.g., validation research vs. solution proposal). In cases where the levels of applicable type facets are relevant, we selected the most fitting category based on objective aspects and discussion within the group.

- **Hypothesis guessing**: since the authors of this article are familiar with systems engineering and SysML, some outcomes might be expected such as increasing publications over the years, or close relationships among research groups. We minimized this risk by using an open research design where we have generated knowledge instead of only checking it.

### 5.4 External validity

The external validity is concerned with “generalization,” and whether the result of a study can be generalized outside the scope of the study or not. According to this validity, there are three main risk types [61]: (i) interaction of participants and treatment, (ii) interaction of environment/setting and treatment, and (iii) interaction of history/timing and treatment. However, in the presented SMS, we do not aim for generalization. Given our scope (keywords, time frame, etc.), the SMS aimed for completeness; however, no extensive literature survey can ever claim to be complete. Our SMS is concerned with scientific research on SysML and can not be generalized to closely related research field. Although some conclusions could be generalized to a broader topic (e.g., lack of evaluation research studies), we did not draw such general conclusions.

### 6 Conclusion and outlook

In this article, we report on our findings regarding the investigated research topics on SysML over the last thirteen years by performing a systematic mapping study. We found out that initially most of the publications were published in systems engineering venues, but since 2013, the research interest on SysML topics moves more toward software engineering. It may be concluded that this moving interest results from the fact that in 2013 Industry 4.0 initiatives started to implement their visions such as CPPS, IoT, IIoT, and others. Therefore, SysML has been very strongly represented in the production application area since that time. Also it seems that the research interest on SysML topics seems to be stronger in Europe than on other continents, since the Industry 4.0 vision started in Germany. However, in Asia and the USA, there started also similar initiatives known under the umbrella “advanced manufacturing” [10] which also stimulate research on software engineering and SysML.

It can be summarized that out of the nine SysML diagram types, the following ones are mainly used: requirement diagram, parametric diagram, activity diagram, state machine
diagram, block definition diagram, and internal block diagram. It turned out that the two newly introduced diagram types—requirement diagram and parametric diagram—are accepted and frequently used by the academia research community. SysML is well established as modeling language for designing, analyzing, and verifying complex systems. However, many researchers customize SysML for their purposes, and therefore, define their own profiles since SysML seems still too generic for some domain-specific tasks (e.g., SysML4Modelica [42,50], SysML4Mechatronics [4,24] to mention just a few approaches). An additional finding is that SysML is lacking of operational semantics. Some approaches aim to overcome this gap such as fSysML [3] which is similar to fUML (a foundational subset of UML for executable UML models).

**Toward SysML v2**
The OMG is currently working on a new version of SysML in version 2 (abbreviated SysML v2). Based on the first insights from the draft SysML v2 Requirements\(^7\), it becomes apparent that the main challenges regarding the usage of SysML, which we have identified and discussed in the presented mapping study, were also admitted in the current work of the standardization group. For instance, SysML v2 is intended to expand the requirement diagram by formal definitions of non-textual requirements in order to make these requirements more general and subject to automated validation. Additionally, the draft addresses the issue of ambiguous operational semantics of SysML, trying to solve this ambiguity similarly to the fUML initiative. There are also planned enhancements to have a timing component in models, which is an important issue, e.g., when modeling continuous systems in combination with discrete systems.

Based on the presented SMS and its results and main findings, we identified the following research directions for future work.

**Research direction 1: life cycle support**
The results show that there is only limited support when using SysML in the implementation phase, and very limited support for describing the whole life cycle of a system from design until operation and back again, for implementing so-called “liquid models” [36]. Therefore, a future research direction is to exploit and adapt SysML for supporting the execution and analysis of systems during runtime and to align operational data with design models.

**Research direction 2: modeling hybrid systems**
Most of the selected publications consider either discrete or continuous challenges when designing systems [6,23]. This means that very rarely hybrid solutions in systems design are taken into account [19]. Therefore, further investigations should be undertaken for defining formal semantics for SysML to close the gap when combining discrete and continuous modeling and simulation.

**Research direction 3: operational semantics for SysML**
Currently, there is no support, e.g., to shift property specification and verification tasks up to the model level. There is still a rule-based operational semantics missing to ensure a step-wise, state-based semantics, e.g., to describe a finite execution trace through a sequence of changes. In this context, a future research direction is to define a rule-based operational semantics for SysML, e.g., based on foundations done in the context of fUML.

**Research direction 4: deeper analysis of the publication corpus for further research questions**
Our result set provides a good foundation for deeper analysis for specific topics related to SysML regarding particular research fields. For example, the contribution category notation can be further differentiated into various language engineering aspects (e.g., profiling, translation to another languages, etc.). Based on such analysis, it is possible to characterize similarities and differences among various approaches.

**Acknowledgements**
Open access funding provided by Johannes Kepler University Linz. This work has been supported by the Austrian Federal Ministry for Digital and Economic Affairs, the National Foundation for Research, Technology and Development. The research reported in this paper has also partly been supported by the FWF in the Project TETRABox under the Grant Number P28519-N31, the Austrian Ministry for Transport, Innovation and Technology, and the Province of Upper Austria in the frame of the COMET center SCCCH.

**Open Access**
This article is distributed under the terms of the Creative Commons Attribution 4.0 International License (http://creativecommons.org/licenses/by/4.0/), which permits unrestricted use, distribution, and reproduction in any medium, provided you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license, and indicate if changes were made.

**A SysML-papers**


of the International Conference of numerical analysis and applied mathematics (ICNAAM 2016), vol. 1863 (2017)


Thirteen years of SysML: A systematic mapping study


 Springer


Giese, H., Hildebrandt, S., Neumann, S.: Model Synchronization at Work: Keeping SysML and AUTOSAR Models Consistent. In: Graph Transformations and Model-Driven Engineering - Essays Dedicated to Manfred Nagl on the Occasion of his 65th Birthday, Lecture Notes in Computer Science,


Gross, J., Mukherjee, R.: Integrating Multibody Simulations With SysML. In: Proceedings of the
13th years of SysML: a systematic mapping study


Thirteen years of SysML: a systematic mapping study


[S257] Hsu, J.: Applying systems modeling language to a simple hardware system.


Thirteen years of SysML: A systematic mapping study


[S354] Liu, Y., Yuan, W., Fan, H., Cao, Y.: Research on information integration framework of SysML based


Muller, M., Roth, M., Lindemann, U.: The hazard analysis profile: Linking safety analysis and SysML. In: Proceedings of the 10th Annual International Sys-
Thirteen years of SysML: A systematic mapping study


Nomsiri, S., Christophe, F., Coantea, E., Mokamel, F.: A combined design structure matrix (DSM) and discrete differential evolution (DDE) approach for knowledge management and design slicing: A case study. Information and Software Technology 54(6), 569–590 (2012). https://doi.org/10.1016/j.infsof.2012.01.005


Thirteen years of SysML: A systematic mapping study


B Books and theses (not used)

SysML-Books


SysML-Theses


References


Publisher’s Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.
Sabine Wolny is a Ph.D. student working in the Christian Doppler Laboratory for Model-Integrated Smart Production (CDL-MINT) at the JKU Linz in the module Reactive Model Repositories. Before she was a doctoral student at the Doctoral College for Cyber-Physical Production Systems (CPPS) at TU Wien. Her topic of interest is SysML-based modeling and execution of systems. In addition to CDL-MINT, she is working since 2013 in the Research Unit of Building Physics at TU Wien with a focus on project management and developing software solutions. For more information, please visit https://www.se.jku.at/sabine-wolny/.

Alexandra Mazak is a senior researcher, heading the module Reactive Model Repositories in the Christian Doppler Laboratory for Model-Integrated Smart Production (CDL-MINT) at the Johannes Kepler University Linz and is also working as a scientific advisor for technology policy in the Austrian Council for Research and Technology Development. Her research interests comprise data integration, statistical modeling and forecast as well as model-driven systems and software engineering in the research field of Industry 4.0. She headed numerous national funded projects in this research field. For more information, please visit https://www.se.jku.at/alexandra-mazak-huemer/.

Christine Carpella was an industrial researcher at SCCH, working primarily in business process and systems modeling. She holds an M.Sc. and an M.A. in Computer Science from the University of Applied Sciences of Upper Austria and a Ph.D. from the Johannes-Kepler-University Linz. Since 2017, she is working at ENGEL AUSTRIA GmbH, a company producing injection molding machines.

Verena Geist holds a Diploma in Software Engineering from the University of Applied Sciences Upper Austria, Hagenberg, and a Ph.D. from the Johannes Kepler University Linz, Austria. Since 2005, she is a researcher at the Software Competence Center Hagenberg (SCCH). Her main research areas are business process modeling, information systems, formal methods, and model-based systems engineering.

Manuel Wimmer is full professor leading the Institute of Business Informatics-Software Engineering at the Johannes Kepler University Linz, and he is the head of the Christian Doppler Laboratory CDL-MINT. His research interests comprise foundations of model engineering techniques as well as their application in domains such as tool interoperability, legacy modeling tool modernization, model versioning and evolution, and industrial engineering. For more information, please visit https://www.se.jku.at/manuel-wimmer/.
Towards Liquid Models: An Evolutionary Modeling Approach

A. Mazak and M. Wimmer;
DOI: 10.1109/CBI.2016.20
Towards Liquid Models: An Evolutionary Modeling Approach

Alexandra Mazak and Manuel Wimmer
Business Informatics Group
Institute of Software Technology and Interactive Systems
TU Wien, Austria
Email: {mazak, wimmer}@big.tuwien.ac.at

Abstract—Today, we recognize a discrepancy between design time models concentrating on the desired behavior of a system and its real world correspondents reflecting deviations taking place at runtime. In order to close this gap, design time models must not be static, but evolutionary artifacts so-called “liquid” models. Such liquid models are the cornerstone of our future research project “CDL-MINT: Model Integrated Smart Production”. In this position paper, we present an early result of this project: the liquid models architecture for linking design models to runtime concerns, which are derived from distributed and heterogeneous systems during operation. We elaborate on the proposed technologies for the respective architecture layers and identify the research challenges ahead.

I. INTRODUCTION

Forecasts show that in the upcoming years most of the things and devices we interact with will be linked to a global computing infrastructure [1]. In the automated manufacturing engineering domain, the Internet of Things (IoT) makes it possible to create networks incorporating the entire manufacturing process to convert production to smart production [2]. The establishment of IoT in manufacturing (also known as Industrial Internet) will have a disruptive impact on the IT industry. This impact will result in the convergence of the physical world and the virtual world in the form of Cyber Physical Production Systems (CPPS) [3]. This represents a new tendency, in which the physical environment is populated by interconnected and communicating objects (e.g., sensors, actuators, and other smart devices) capable of autonomously interacting with each other and with the environment itself. As a result, both, the volume and the level of detail of the corporate data generated will highly increase.

Due to this ever growing importance of flexibility, also resulting from shorter innovation cycles, rapidly changing customer needs, etc., the role of software is becoming more and more important in the industrial automation domain. Especially with the transition in the next industrial revolution, in Germany and Austria known as Industrie 4.0 [4], several new challenges arise. As a consequence systems may no longer be designed to stay for decades in certain settings, but they may be designed to evolve and exist in different variants adapted to certain contexts [4]. Particularly in Industrie 4.0 scenarios, there will be an extended usage of models also during operation time that allows runtime monitoring and design model enhancement [2], [5], [6], but still, models are completely isolated from the running systems which have been beforehand described by these models. Generally, the automated manufacturing engineering domain exploit the benefits of modeling for code generation [7]. Thereby, the dynamic extraction of runtime models to better link operation with engineering is often overlooked. However, current modeling foundations and practices are lacking behind this emerging requirement. Models are still considered as static entities, basically neglected in later lifecycle phases of systems.

One reason is that models are never complete and only created for a specific purpose, such as being a “blueprint” of a system to be developed [8]. Since, modelers or system engineers tend to concentrate on the desired behavior of the system during its design, they are not aware of the many deviations that may take place at runtime. These deviations may result in discrepancies between the design model and its real world correspondent. This gap can be seen as a discrepancy between models created at design time and real operations within systems. In software engineering this gap is referred to as the DevOps gap. DevOps is a trend ensuring a continuous feedback loop (i.e., “model in the loop”) between development (Dev) and operations (Ops) [9]. DevOps aims at a better integration of all activities in software development and the operation of an application system lifecycle in order to continuously deploy stable versions of application systems [10].

If models should be fully integrated in the lifecycle of a system, having only an a-priori description of the system as a product of the initial engineering phases is not enough. From this perspective models are no longer static descriptions, but evolutionary artifacts so called “liquid models”. The term “liquid” is used to stress that models as any kind of artifact should not be isolated and frozen, but reusable and evolutionary. The evolutionary aspect of engineering artifacts refers to the fact that they change over time. In engineering processes models are generally developed from initial ideas to first drafts. They are then continuously revised, often by taking
Liquid models represent a cornerstone of our research project CDL-MINT: Model-Integrated Smart Production. Thus, this project aims at linking design models to runtime concerns which are derived from distributed and heterogeneous systems during operation. As a first step in this direction we develop a liquid models architecture that forms the core of this paper. This architecture comprises three layers on top of heterogeneous data sources. We discuss the goals of each layer and the potential technologies to be used. Furthermore, we identify the research challenges ahead.

The remainder of this paper is structured as follows: In Section 2 we introduce related work we considered in the development of our architecture. We discuss the importance of model repositories. Emerging approaches in the field of process mining in the area of workflow systems and runtime models in the area of model-driven engineering (MDE) [11] are noticed as first approaches considering systems in operation. Data analytics techniques are evidently required to learn from systems in operation. Given the state-of-the-art, we identify key research issues towards liquid models in Section 3. Section 4 introduces our liquid models architecture and discusses its layers in detail. Finally, Section 5 summarizes this paper.

II. RELATED WORK

Several lines of research are relevant to realize our goal to establish liquid models managing operational models that are reactive with respect to their origin (i.e., built from real world data acquired from system operations). In the following, we outline the state-of-the-art of the most relevant ones. First, we discuss emerging model repositories as well as modeling approaches considering runtime aspects in addition to design. Second, we survey techniques to deal with operational data and to turn it into abstracted model representations. In this context, we specifically highlight process mining, runtime models, and data analytics techniques.

A. Model Repositories

Research concerning model repositories comprises mainly two areas: concurrent modeling using a central repository to coordinate the editing of models [12, 13] and scalability in storing and retrieving models [14]. Currently, the general services offered by a model repository are twofold: (i) load a complete model from a repository, and (ii), store a complete model to a repository. Other services, such as more fine grained model loading or manipulation is currently missing in most repositories [15]. The scalability problems of loading large models represented by XML-based documents into memory has been already recognized several years ago. One of the first improved solutions for models is the Connected Data Objects (CDO)\(^1\) model repository, which enables to store models in all kinds of database back-ends, such as traditional relational databases or emerging NoSQL databases. CDO supports the ability to store and access large-sized models due to the transparent loading of single objects on demand and caching them. If objects are no longer referenced, they are automatically garbage collected. There are also several projects for storing very large models, like MongoDB\(^2\) and Morsa [16]. Both approaches are built on top of MongoDB. Furthermore, graph-based databases as well as map-based databases are also exploited for model storage, such as done in Neo4EMF [17, 18] where also different unloading strategies for partial models are explored [19]. In [20], Claesen et al. elaborate on strategies for storing models in a distributed manner by horizontal and vertical partitioning in Cloud environments. A similar idea is explored in [21] where different automatic partitioning algorithms are discussed for graph-based models.

What all the mentioned approaches have in common is that models are residing behind the walls of the model repository without a proper connection to the environment as is needed, for instance, to provide reactivity by observing runtime environments during operations.

This is one particular goal of our approach to integrate model streaming to model repositories in addition to full loading and storing of complete models in order to have means for observing systems during operation.

B. Runtime Models

There are several different approaches for runtime modeling. All of them aim on bridging the gap between design time modeling and runtime modeling to enable runtime analysis. Blair et al. [22] show the importance of supporting runtime adaptations to extend the use of model-driven engineering. They propose models that provide abstractions of systems during runtime. These operational models are an abstraction of runtime states. Due to this abstraction, different stakeholders can use the models in various ways, like dynamic state monitoring or observing runtime behavior. Hartmann et al. [23] go one step further. They combine the ideas of runtime models with reactive programming and peer-to-peer distribution. Reactive programming aims on enabling support for interactive applications, which react on events by focusing on streams. For this purpose a typical publish/subscribe pattern, well known as the observer pattern in software engineering [24], is used. Khare et al. show the application of such an approach in the IoT domain in [25].

Luckham [26] introduces Complex Event Processing (CEP) by defining complex events which are correlated among each others. Such an approach of CEP on stream data was described by Saleh et al. [27]. Hartmann et al. [23] define runtime models as a stream of model chunks, like it is common in reactive programming. The models are continuously updated during runtime, therefore they grow indefinitely. With their interpretation that every chunk has the data of one model element, they process them piecewise without looking at the total size.

\(^1\)http://code.google.com/a/eclipselabs.org/p/mongo-emf
\(^2\)http://projects.eclipse.org/projects/modeling.xml.cdo

105

Authorized licensed use limited to: TU Wien Bibliothek. Downloaded on August 28,2021 at 13:16:28 UTC from IEEE Xplore. Restrictions apply.
In order to prevent the exchange of full runtime models, peer-to-peer distribution is used between nodes to exchange model chunks. In addition, automatic reload mechanism are used to respond on events for enabling reactive modeling. As the models are distributed, operations like transformations have to be adapted. For this purpose transformations on streams as proposed by Cuadrado et al. [28] can be used.

In our approach, we will explore this path of research even further by closely combining model streaming reasoning, reactive programming, and operational data monitoring. For this purpose we have to integrate powerful reasoning algorithms which may be inspired from the fields of process mining in particular and of data analytics in general.

C. Process Mining

Process mining (PM) is a process-centric management technique bridging the gap between data mining and traditional model-driven Business Process Management (BPM) [29], [30]. In this field of research business processes are analyzed on the basis of event logs. Events are defined as process steps and event logs as sequential events recorded by an information system [31]. This demonstrates that unlike BPM approaches PM works on the basis of event data instead of designed models. The main challenge is to capture behavioral aspects. In [29], van der Aalst introduces specialized algorithms (e.g., the ω-algorithm) to extract knowledge from event logs. Therefore, an algorithm produces a Petri net, which can easily be converted into a process model as, for instance, a BPMN model, or a UML activity diagram.

A process model is used in the (re)design, configuration and implementation phase, whereas, data provides insight on actual processes for monitoring and diagnosis purposes [32]. The main objective of PM is to extract valuable, process-related information from event logs for providing detailed information about actual processes, for instance, to identify bottlenecks, to anticipate problems, to record policy violations, to streamline processes, etc. [30]. PM is not limited to this control-flow perspective. There are further perspectives as introduced in [29], namely (i) the organizational perspective focusing on information about resources (e.g., people, systems, departments) hidden in event-logs, (ii) the case perspective focusing on properties of cases, and (iii) the time perspective concerned with the timing and frequency of events.

In [29], van der Aalst lists three basic PM goals, which are (i) discovery, (ii) conformance, and (iii) enhancement. Discovery means to take an event log as input and to produce a process model as output. When targeting for conformance of an existing process model is compared with an event log of the same process. This means an event log and a model are used as input and a diagnostic information is produced as output. Thereby, a user can check whether information recorded in the log conforms to the intended model and vice versa. Conformance checking can be applied to any kind of models (e.g., business process models, declarative process models, etc.). The third type of PM is called enhancement. Its idea is to improve or extend an existing model. It takes an event log and a model as input and produces a new model as output.

Current event processing technologies usually monitor single streams of events at a time. Even if users monitor multiple streams, they often end up with multiple "valo" views. A more unified view is needed that correlates with events from multiple data streams of various sources and in different formats. Thereby, heterogeneity and incompleteness of data are major challenges [33]. Mostly, PM operates on the basis of events that belongs to cases that are already completed [34]. This off-line analysis is not suitable for cases which are still in the pipeline. In [29], the author mixes current data with historic data to support on-line and off-line analysis.

D. Data Analytics

Data analytics deals with the acquisition of information derived from a big amount of data. Its target is to recognize new models within these quantities of data, to recover existing patterns and to discover new patterns [35]. Therefore, complex analytical procedures and methods of various fields, e.g., machine learning, data mining, statistics and mathematics are being applied [36].

In [37], Fayyad et al. define data mining as the process of knowledge discovery in databases. This process consists of several steps, starting with the selection, the pre-processing, the transformation, the mining, the interpretation and the evaluation of data. Various methods can be applied that deal with the application of suitable models on observed data and the finding of interesting patterns within these observed data [38]. There are statistical models that permit non-deterministic and deterministic effects, and logic approaches [37]. These methods are commonly based on algorithms known from other areas, as for instance, machine learning.

Machine learning deals with the construction of systems that optimize performance criteria based on sample data or past experiences [39]. Usually these systems build on models that are "trained" on the behavior of existing data with the help of machine learning algorithms. These models can be differentiated into descriptive and predictive models. Descriptive models attempt to learn new information out of existing data, whereas predictive models provide predictions on the future. Additionally, machine learning can be subdivided into supervised and unsupervised learning techniques. Methods for supervised learning attempt to learn a hypotheses based on known data (target value and / or value for reward), whereas unsupervised learning applies algorithms that do not need to known target values or values for reward beforehand [40].

There are many existing research projects focusing on machine learning in the field of Big Data analytics [36]. For instance, the WEKA toolkit provides various machine learning algorithms for pre-processing, classification, regression, clustering, and the visualization of data. The project Apache Mahout6 works on the efficient and parallelized

6http://www.cs.waikato.ac.nz/ml/weka
7http://mahout.apache.org
implementation of machine learning algorithms and provides these algorithms open source to the community. Additionally, the MLlib machine learning library implemented in the course of the Apache Spark project provides a broad set of machine learning algorithms, which can easily be applied on a large amount of data offering high-performance.

Additionally to this overview, there are various techniques that can be applied in order to learn new models or improve existing ones. These techniques primarily rely on statistical models and algorithms of the research fields of neuronal networks as well as genetic algorithms [35]. As concrete research lines for our approach, we combine these techniques with model repositories in order to provide scalable solutions for liquid models which may have to deal with huge amounts of operational data. One important open challenge, for linking design models to runtime models, is how to combine advanced data analytics technologies and modeling technologies, such as model transformation engines or model checkers, in order to get the best of both worlds. This is a particular challenge towards a moving target, since there are still several open challenges in these areas to be combined.

III. OPEN RESEARCH ISSUES

This section summarizes key research issues (RI) we have identified in the state-of-the-art to realize our research goals.

A. Real-time Analytic Correlation across Multiple Data Streams

Manufacturing machines and sensors continuously generate and transmit data (e.g., detailed logs) about their actions and current condition. As a result, there is an exponential growth of volumes of unstructured, semi-structured, and multi-structured data (e.g., machine data, sensor data, event streams, XML, CSV, SCADA). Data streams are ordered and potentially unbounded sequences of data points created by a typically non-stationary data generating process. However, traditional algorithms for data mining cannot mine even a fraction of these streams in real-time. The situation is aggravated by the fact that data is distributed over a variety of different sources in various latencies (from batch to real-time). Merging this data tend to be problematic, e.g., caused by heterogeneities (e.g., technical, syntactical, semantical), different levels of granularity, etc. A further challenge is to deal with incomplete and noisy data streams as well as with concept drifts (i.e., changes on data streams while being observed). For analytical purpose (i) each event has to be captured from a stream, (ii) events of interest have to be separated from noise, (iii) correlations with other streams and databases have to be established, (iv) it has to be reacted to the events of interest in real-time, (v) and events have to be stored in an appropriate model structure for on-line and off-line analytics. In order to tackle this research issue, we see two sub research issues that have to be addressed:

RI 1.1: How to integrate distributed and heterogeneous data streams?

RI 1.2: How to provide reactive model stream processing mechanisms within existing modeling technologies and accompanying languages?

B. DevOps-centric Methodology to support Evolutionary Model Mining

DevOps is a trend ensuring a continuous feedback loop between development (Dev) and operations (Ops) [9] concerning technical as well as social aspects. DevOps aims at a better integration of all activities in software development (e.g., also in the context of MDE) and the operation of an application lifecycle in order to continuously deploy stable versions of applications [10]. The monitoring process is, among others, one of the key factors for a successful implementation of DevOps. In our research project, we transfer the DevOps concept to the industrial automation domain for a continuous end-to-end engineering in this ever-changing environment. In particular, by having integrated and unified model streams, model mining techniques can be applied to extract operational models which can be matched against the design models in order to enhance them with more knowledge from observations during operation time. In order to tackle this research issue, we see two sub research issues that have to be addressed:

RI 2.1: How to realize scalable model mining techniques on top of model streams?

RI 2.2: How to propagate back and capture the observed knowledge from operation to design models?

IV. LIQUID MODELS ARCHITECTURE

In this section, we lay out our envisioned architecture (cf. Fig. 1) to tackle the aforementioned research issues.

A. Overview

The full integration of design models in the systems’ lifecycles requires a continuously acquisition of real-time data (e.g., machine data, sensor data, event streams) from various distributed and heterogeneous systems. As a first step in our approach, we acquire heterogeneous event logs from various data sources. This kind of data acquisition is taking place during operation time in the various runtime environments. Therefore, the data is not a finite set, but, a continuous stream of heterogeneous data. After resolving certain forms of heterogeneity (e.g., technical, syntactical, semantical), as discussed in [41], [42], the data is processed within distributed operational models. These operational models implement statistical methods as well as machine learning algorithms, targeting the identification of patterns and anomalies. Based on this runtime information, the models created during design time can constantly be improved. The alignment between operational models and design models is what we call the DevOps bridge in our architecture.

Fig. 1 gives an overview of the liquid models architecture. We describe this figure bottom-up by a 4-layered stack. Level 0 (L0) represents distributed, physical components, sending data streams during operation. Due to their heterogeneity, these data streams have to be aligned and synchronized. Traversing
the layers bottom-up (L0-L3), heterogeneities, distribution and timing inconsistencies are resolved. At the top layer (L3) the operational models are built, based on model mining from subscribed topics as for instance non-functional properties [43]. Hereby, a topic is an orchestration of data streams with the application of potential filters. Even if only filtered topics of interest are subscribed by operational models, the data volume can become very high. Therefore, we propose a distributed modeling approach in order to ensure scalability and availability. Please note that the model repositories embedded in the outlined architecture of Fig. 1 become reactive model repositories. First, the operational models are reactive with respect to event occurrences in the runtime environments. Second, the design models are reactive with respect to changes in the operational models. Finally, there are the design models at the top of Fig. 1, which we assume as given artifacts.

In the following subsections, we outline how we will realize this architecture, namely, how to integrate distributed and heterogeneous data streams, how to react on important events in unified model streams, how to realize distributed model repositories, and finally, how to apply model learning on operational data to bridge the DevOps gap mentioned before.

### B. Level 0: Heterogeneous Data Sources

The starting point of our approach are data streams from various and physically distributed data sources (cf. Fig. 1, level L0 $DS_1, ..., DS_n$). This sources have to be processed in real-time, where there is no random access and which can be read only once a time (e.g., readings from sensor networks). These data streams are infinite sequences of ongoing events. They are received continuously and in real-time, either implicitly ordered by arrival time, or explicitly associated with times-
tamps. Nearly everything can be considered as a stream, e.g., sensor data, machine data, user inputs, calculation results (see Fig. 1, ground level). The data content is scattered over several sources, which are based on different technologies (e.g., CoAP, OPC UA, MQTT, FTTE, OSLC, etc.). In order to enable real-time data handling, we provide appropriate adapters for further processing the data. These adapters are used to resolve technical and data model heterogeneities in order to create isolated profiles for each source stored in a repository. Based on our previous work introduced in [44], [55], we provide mapping operators to resolve structural heterogeneity and data fusion operators to deal with duplicates and conflicts. Additionally, we aim to develop an algorithm that automatically records all information about the integration task.

C. Level 1: Data Streaming
In a next step, the data streams are the input for the Data Distributed Service (DDS) [46] (cf. Fig. 1, orchestraion and filtering on L1) which is a proven international standard from the Object Management Group (OMG). The DDS is a middleware protocol and API standard for data-centric connectivity to save data communication overhead. It provides a data-centric solution to understand the schema of shared data. We select DDS as a first potential candidate to support our approach. However, we will also critically evaluate whether there are better alternatives for certain scenarios. DDS allows to filter the data that is actually needed. For this purpose DDS offers communication by publishing and subscribing to topics for collaborative filtering (cf. Fig. 1, L2). Subscriptions can specify time and context filters to get specific subsets of the data being published on the topic. In our approach, we use DDS for filtering and smoothing the time component, to get the right data at the right place at the right time.

The filtering mechanisms help to reduce the volume of observed data streams. Nevertheless, the filtered streams are still a continuous and infinite sequence of data. There are various methods for accessing information provided by streams, as for instance, Complex Event Processing (CEP) [27] or C-SPARQL [47]. Compared to these emerging techniques, classical DDS stream reasoning techniques seems limited and have to be extended in this respect. In a first step, we plan to investigate the appropriateness of different technologies for model streaming. For example, we may investigate C-SPARQL as continuous query language that enables stream reasoning over RDF data streams [47], [48]. An RDF data stream is not static, it is continuously produced and with a timestamp annotated RDF triple identified by an Internationalized Resource Identifier (IRI) [48]. The windows function enables to extract certain criteria over these infinite sets. This extraction can be a given number of triples or a variable number of triples occurring in a given time interval. By using continuous query languages these triples can be queried in order to get unified data streams for further processing. In order to allow the reuse of existing model query and transformation languages, we plan to integrate concepts of C-SPARQL within the Object Query Language (OCL) to allow an easier adoption of stream reasoning for engineers compared to switching to the RDF level. Initial experiences for implementing approximate model transformations may be reused where we already reformulated the classical OCL operators for potentially infinite models, introduced in our work in [49].

D. Level 2: Model Streaming
The unified streams provide a “dynamic picture” based on various DDS-topics (e.g., non-functional properties). However, due to the still vast amount of data, these streams cannot be stored in memory. Thus, we use statistical methods to generate inductive-empirical models to extract operational models which can be matched against the design models (cf. Fig. 1, L3). Thereby, we aim for the development of a hybrid approach by combining techniques and methods of advanced process mining, multivariate statistics, stochastic and complex event processing. The challenge will be the efficient combination of these techniques to extract more insights from data streams. The idea of a hybrid approach is demonstrated by a short example: to mine unified data streams for classification, we combine Online Convex Programming (OCP) and Support Vector Machines (SVM). OCP is a general framework for online prediction algorithms [50] and SVM is a vector-based machine learning technique [40]. By following this approach not all data streams must be kept in memory. Thereby, optimal event-based feature vectors can still be calculated by means of optimizing a convex function on a convex set. This convex set may be generated by techniques adapted from CEP. This is one candidate method for classification, there may be others which have to be identified and investigated in the course of our future work.

In the context of data (stream) mining the “curse of dimensionality” is a critical challenge, too. In order to overcome this problem, we apply statistical methods for dimension reduction (e.g., Principal Component Analysis, or data stream clustering by micro-clustering, or the BIRCH algorithm, as well as canonical analysis techniques) combined with methods of CEP (e.g., sliding time window) and approaches for combinatorial optimization. Depending on certain scenarios, we apply advanced time series analysis, machine learning techniques such as neural networks, genetic algorithms, and stochastic models (e.g., Hidden Markov Models) to generate appropriate operational models. For the fulfillment of quality and appropriateness of the outcomes, we develop a metric to evaluate which algorithms are best suited, since some algorithms may be more challenging to adapt than others.

E. Level 3: Operational Models
For persisting the operational models, we plan to implement distributed model repositories in the spirit of Hartmann et al. [23]. The models within these repositories subscribe on topics delivered by DDS. The events within the topics are captured, unified and saved in the reactive operational models. In this way the semantic heterogeneity is resolved. For transformations between different operational models (cf. Fig. 1, ground level).
dotted arrow between the Distributed Operational Models), we plan to built on existing transformation languages like ATL and approximative transformations as presented in our work in [49].

The operational models build the starting point for the DevOps Bridge. Being in a “data-rich” situation we divide the model samples into three parts, a training set, a validation set, and a test set. For model (stream) learning, we use the training set to fit the models, the validation set to estimate prediction error for model selection, and the test set to assess the generalization error of the finally chosen model [40]. For training purpose, algorithms need to access the complete training set several times. Providing this access is not straight forward, since data streams deliver constantly new data. Additionally, it is important to evaluate how well the algorithm is able to adapt to changes during a learning phase, known as concept drift [30]. For instance, Aggarwal et al. [51] introduced an approach where data streams are splitted into consecutive horizons to overcome the concept drift problem during clustering. Another example could be to calculate the probability when changes had happened by time series analysis in combination with CEP techniques. Change detection may be used as filter to receive all changes within time slots. This approach requires an appropriate classifier. For example, the Kulman filter (limited to multi-dimensional Gaussian distribution) may serve as a suitable classifier in case of time series analysis with Hidden Markov Models [52]. In addition, clustering techniques applied on time series help to identify the internal and external triggers that might have caused changes.

The validation step can either be conducted analytically (Akaike information criterion, Bayesian information criterion), or by efficient sample reuse (cross-validation and bootstrap) [53]. The assessment of the generalization performance is extremely important in stream learning [54]. It shows the interplay between bias, variance, and model complexity. It guides users in model selection (i.e., when estimating the performance of different models in order to choose the approximate best one) and in model assessment (i.e., when estimating the prediction error of the chosen model) [40]. In a final step, we adapt the conformance checking technique, as introduced in [29], to align operational models and design models.

This alignment includes the extension of design models with information about derived information from operational models, as well as information about runtime characteristics unknown at design time. For the latter, standard languages will be used, such as for instance the OMG MARTE profile [55] for capturing real-time and embedded characteristics of the operational systems (e.g., resource allocations and performance characteristics) or existing languages to represent variants of a system and enrich the variant description with important operational information such as performance, reliability, and responsiveness. In the model learning task, we will enrich the conformance checking method introduced in [29] with techniques from supervised and unsupervised learning and concepts of non-functional property languages, to repair design models that are not aligned well with reality, which provides a basis to close the gap between DevOps.

V. CONCLUSION

In this position paper, we present several upcoming research challenges for stimulating a shift from isolated, one-shot, monolithic system descriptions to evolutionary, reusable predictions. We focus on how to connect runtime environments to model repositories to extract operational models and their connection to design models. Thus, specific techniques are needed to connect to runtime environments and to deal with model streams to efficiently react to events occurring in various highly physically distributed data sources at the ground level.

We present an architecture for this purpose that uses appropriate techniques on four dedicated levels. Between level L0 and L1, we adapt an idea from previous work [44], [45] applying adapters to resolve technical and data model heterogeneities in order to create isolated profiles for each source stored in a repository. In a next step, between L1 and L2, we introduce Data Distributed Services (DDS) to filter these profiles by publishing and subscribing to topics in order to reduce the volume of observed data streams. At this level, we present additional methods (CEP, C-SPARQU) for stream reasoning and for enhancing the DDS technique. Between L2 and L3, we outline some promising statistical methods to generate inductive empirical models. These kind of operational models are then matched to the corresponding design models in order to improve the latter ones. Thereby, design models are extended with information previously unknown at design time.

REFERENCES

genbau e. V. (VDMA), Zentralverband Elektrizität- und Elektronikindustrie e. V. (ZVEI), Ergebnisbericht der Plattform Industrie 4.0, 2015.
8 Towards Liquid Models: An Evolutionary Modeling Approach


9 Model-Driven Time-Series Analytics

S. Wolny, A. Mazak, M. Wimmer, R. Konlechner and G. Kappel;
DOI: 10.18417/emisa.si.hcm.19
Model-Driven Time-Series Analytics

Sabine Wolny*, a, Alexandra Mazak a, Manuel Wimmer a, Rafael Konlechner a, Gerti Kappel a

* Corresponding author.
E-mail: wolny@big.tuwien.ac.at

This work has been supported by the National Foundation for Research, Technology and Development (CDG), the Austrian Federal Ministry of Science, Research, and Economy (BMWFV Austria), and the TU Wien research funds.

Abstract. Tackling the challenge of managing the full life-cycle of systems requires a well-defined mix of approaches. While in the early phases model-driven approaches are frequently used to design systems, in the later phases data-driven approaches are used to reason on different key performance indicators of systems under operation. This immediately poses the question how operational data can be mapped back to design models to evaluate existing designs and to reason about future re-designs. In this paper, we present a novel approach for harmonizing model-driven and data-driven approaches. In particular, we introduce an architecture for time-series data management to analyse runtime properties of systems which is derived from design models. Having this systematic generation of time-series data management opens the door to analyse data through design models. We show how such data analytics is specified for modelling languages using standard metamodelling techniques and technologies.

Keywords. Model-Driven Engineering • Time-Series • Data Analytics • Language Engineering

1 Introduction

In model-driven engineering (MDE), models are the central artefact and used as a main driver throughout the software development process, finally leading to an automated generation of software systems (Lara et al. 2015). In the current state-of-practice in MDE (Brambilla et al. 2017; Karagiannis et al. 2016), models are used as an abstraction and generalization of a system to be developed. By definition, a model never describes reality in its entirety, rather it describes a scope of reality for a certain purpose in a given context (Brambilla et al. 2017). Thus, models are mostly used as prescriptive models for creating a software system (Heldal et al. 2016). Such design models determine the scope and details of a domain of interest to be studied. For this purpose, different types of general modelling languages (e.g., state charts, class diagrams, etc.) may be used or domain-specific languages (DSLs) (Karagiannis et al. 2016) may be employed. It has to be emphasized that engineers typically have the desirable behaviour in mind when creating a system, since they are not aware in these early phases of many deviations that may take place at runtime (van der Aalst 2016).

According to Brambilla et al. (2017) the implementation phase deals with the mapping of prescriptive models to some executable systems and consists of three levels: (i) the modelling level where the models are defined, (ii) the realization level where the solutions are implemented through artefacts that are used in the running system, and (iii) the automation level where mappings from the modelling to the realization phase are made. However, these levels are currently only used for down-stream processes. The possibility of up-stream processes is mostly neglected in MDE (Mazak and Wimmer 2016). Especially, for later phases of the system lifecycle descriptive
models may be employed to better understand how
the system is actually realized and how it is operat-
ing in a certain environment (Mazak and Wimmer
2016). Compared to prescriptive models, those
descriptive models are only marginal explored in
the field of MDE, and if used at all, they are built
manually.

In this paper, we move towards a well-defined
mix of approaches to better manage the full life-
cycle of systems by combining prescriptive and
descriptive model types. In particular, we intro-
duce a model-driven time-series data analytics
architecture for harmonizing model-driven and
data-driven approaches. Based on this architec-
ture, we show how data analytics can be specified
for modelling languages using standard metamod-
ing techniques. This means, design-oriented
languages are equipped with extensions for repres-
enting runtime states as well as runtime histories,
which in turn allow the formulation and com-
putation of runtime properties with the Object
Constraint Language (OCL). This approach has
the advantage to directly interpret measurements
and events within the design models without in-
troducing an impedance mismatch.

The remainder of this paper is structured as
follows. Section 2 provides the background for
this paper by introducing a motivating example
which is subsequently used as running example.
Section 3 gives an overview of our architecture
for unifying model-driven and data-driven ap-
proaches. In Section 4, we present in detail how
time-series analytics can be integrated in metamod-
els. Section 5 discusses the related work. Finally,
in Section 6, we conclude with an outlook on
future work.

2 Motivating Example

In this section, we introduce a motivating example,
which will subsequently become the running ex-
ample of this paper. We first describe the example
from the modelling perspective, then from the
realization perspective with a focus on runtime
data collection, and finally conclude with the chal-
lenges we aim to address with this paper.

Model-Driven Perspective

As our motivating example, we consider a
grip-arm robot (gripper) with different position
properties of axis angles: BasePosition (BP), MainArmPosition (MAP), and
GripperPosition (GP). From a device point of
view (cf. Figure 1(a)), the structure of the gripper
component and its behaviour are modelled at
design time by a subset of a SysML-like language,
i.e., blocks with associated state machines. The
top of Figure 1(a) shows the specific properties
(BP, MAP, GP) of the block, whereas the actual
property values depend on the different states
e.g., Idle, Pick Up). The states are given at the
bottom of Figure 1(a).

By the given state machine, property value
changes are modelled. The gripper has certain
positions at initialization, in state Idle and in state
Pick Up. The assumption of the modelled state
machine is that as soon these states are reached,
the position values are set. However, such state
machines are a kind of black box, where only
the discrete values before entering the state and
after leaving the state are known (cf. Figure 1(b)).
While this may be sufficient for several design
tasks and discrete systems, for continuous systems
more information may be required. This is in
particular true for our example case. The gripper
represents a continuous system, since it does not
immediately realize the next position, but needs
time to move to the given place. Usually, such
information is not directly given in a design model,
but it may be important for several tasks such as
optimization, validation, and verification. The
ability to observe property value changes over
time within states may contribute to capture the
current capabilities and shortcomings of systems.
Thus, the presented approach of this paper aims to
transform the black box into a so-called “grey box”
to make the effects of value changes visible (see
Figure 1(c)). For instance, observation sequences
of property value changes are an important base
information of a system’s operation to compute
operating figures to check if the behaviour of each
gripper’s axis corresponds to the defined one in the design model.

**Data-Driven Perspective**

For the technical realization of our example, we developed a simulation model of the gripper consisting of three angle sensors, which we executed by the open source tool Blender\(^1\). We deploy the scenario of a pick-and-place unit, where the gripper picks up different color-coded work pieces, place them on a test rig, picks the items up again and puts them down, depending on their red or green color, in two different storage boxes. The simulation environment receives its commands via Message Queue Telemetry Transport (MQTT) from a server controller implemented with Kotlin\(^2\). The simulation enables to acquire transient data streams in real-time from the angle axes of the gripper (BP, MAP, GP, unit is radian), which are equipped with sensors.

To react on events of interest provided by these data streams, we employ the publish/subscribe pattern. In our example, we subscribe to the sensor topic to receive in a temporal distance of 15 milliseconds the filtered data streams of the sensors of the gripper during simulation. Thereby, we are interested in property value changes (i.e., positions of the axes) in the simulation at given points in time. Messages from the sensor topic are defined in JSON\(^3\) specifying the sending unit as well as the measured data. The following example shows such a message from the angle sensors of the gripper to the controller.

```
{"entity": "GripperArm",
"basePosition": 0.0,
"mainArmPosition": 0.0,
"gripperPosition": 0.0}
```

This example shows the positions of the angle axes at system initialization (see Figure 1). The angle position of each axis has the value 0.0. The default range of the angle values is \([-\pi, \pi]\). To analyze our scenario, it is important to save the measured data over time. For this purpose, we use the time series database InfluxDB\(^4\). InfluxDB allows us to store a large amount of time-stamped data. In addition, by the tool Grafana\(^5\) we can visualize our stored sensor values.

**Challenges**

Our motivating example is discussed from two angles: (i) from the model-driven, i.e., how the intended system should work, and (ii) from the data-driven, i.e., how measurements can be taken from the running system to reason about the actual realization. While the first perspective is lacking concepts to define runtime data such as

---

1. https://www.blender.org
2. https://kotlinlang.org
4. https://www.influxdata.com
5. https://grafana.com
time-series, the second perspective has to correctly interpret the collected measurements. The challenge is how to overcome the gap between those two perspectives (i) to monitor important data from operation, (ii) to align the measurements with the design model in order to provide a semantic anchoring of the data, and (iii) to provide meaningful analytics whereas the results of the analytics are interpretable for the given design models to reason about improvements or fulfilments of given requirements.

3 Unifying Architecture for Model-Driven and Data-Driven Approaches

In order to allow a smooth integration of model-driven and data-driven approaches, we present in this section an architecture, which builds on the classical model to system downstream in terms of code generators, but at the same time, supports an upstream in terms of mapping data back to design models. Figure 2 gives an overview of this architecture. In the following section, a more detailed description of the different parts will be presented based on our running example.

The proposed architecture consists of four main parts. First, the left hand side of Figure 2 captures the classical downstream MDE approach (cf. (a) in Figure 2). At the metamodel layer, the design language is defined with the help of a metamodelling language (in our setting Ecore). Conforming to the design language, the design models are defined at the model level describing the static (i.e., structure) and dynamic aspects (i.e., behaviour) of a system to be developed. For the vertical transition from the modelling to the realization level we assume the existence of model-to-text transformations for code generation. Thus, this part of our architecture describes how we can derive the executable system from the design model as is the state-of-the-art in MDE.

Second, we continue with defining the first part of the up-stream process of runtime data to the design model (cf. (b) in Figure 2). In addition to the actual systems, the runtime observer is generated out of the design model. The runtime observer collects important information from the running system to represent the current state of the system. Those observations should not only be recorded by observing the running system, but should be also representable at the model level. Thus, we extend the design language with a dedicated runtime language. This metamodel defines the syntax to represent snapshots of the running system connected to the design model elements. Those snapshots are represented in the runtime state models which extend the design models and may be directly updated by the runtime observer during runtime. In summary, this part of our architecture maps runtime data at the model level for one single point in time and may be used to monitor a system on the model level.

Third, we define the runtime history of a system (cf. (c) in Figure 2). For reasoning about, e.g., property value distributions, it is important to have the complete history of value changes as starting point as one snapshot is definitely not sufficient for such computations. Thus, in the time series database the observations of the running system are stored. Based on these collected observations, the runtime history models may be directly updated. These models conform to the runtime history language, which is an extension of the runtime language. In the runtime history language, the syntax is defined for representing histories of runtime phenomena of interest, e.g., property values, events, etc.

Finally, after defining those concepts for storing observation histories at the model level, it is also possible to analyse the stored observations (cf. (d) in Figure 2). For this purpose, we define runtime properties based on the Object Constraint Language (OCL) by introducing derived properties for the metamodel elements. These derived properties enable us, e.g., (i) to compute descriptive statistics, (ii) to evaluate monotony behaviour of value changes, or (iii) to compute lower and upper bounds of properties to mention just a few examples. Based on the runtime properties, the runtime property values are computed by analysing the collected time-series. Thus, runtime
data is back propagated to the design models and this mapping allows to interpret the data through the design model elements as there is a clear traceability guaranteed from design elements, runtime states, and runtime histories.

By this architecture, we are able to harmonize model-driven and data-driven approaches, where time-series data management of systems at runtime can be derived from initial design models and be used again at the model layer by importing the time-series to model structures. How this model structures are defined is the topic of the next section.

4 Metamodelling Blueprint for Enhancing Models with Time-Series Analysis

Based on our running example, we further detail in this section how the afore presented architecture can be realized for a given language. In particular, we show for the introduced design modeling language, how the extensions for runtime states, runtime histories, and runtime properties are defined as reusable metamodelling blueprints. The time-series analysis we are focusing on for demonstration purposes is about property value changes of the axis angles (i. e., BP, MAP, GP) of the gripper in our running example.

Design Elements

As already mentioned before, our starting point is the availability of a design modelling language expressed in Ecore. For our running example, we model the structure of the gripper with its properties as a kind of block diagram similar to what is known from SysML. A block has an associated state machine, where different states and transitions are defined. For states, assignments can be defined, which are executed when a state is activated. The assignments in our exemplary language are simple value assignments for the properties of a block. The resulting metamodel for the described design language is shown in Figure 3.

Runtime States

In order to express concrete runtime states on the model level, the metamodel has to be extended with runtime concepts. For this task, there are several existing approaches available, e. g., (Engels et al. 2000; Mayerhofer et al. 2013; Meyers et al. 2014). Most of them add additional metamodelling elements to the design language to describe what
runtime phenomena are of interest and how they are connected to design concepts. For our running example, the runtime language is considered as an extension of the design metamodel to allow representing property values for a given point in time (i.e., for a snapshot of the running system). In addition, transitions may fire during runtime. Thus, the concept of transition firing is introduced. While values are considered by measurements during the operation phase, the firing of transitions are categorized as events. Please note that the relation to the design concepts has to be clearly stated by the runtime concepts, e.g., the value concept is related to the property concept. Figure 4 captures the concrete realization of the runtime extension for our design language.

Figure 3: Design metamodel for the running example.

Figure 4: Runtime metamodel for the running example.

Runtime Histories
To reason about operation figures going beyond one snapshot in time such as distributions, upper and lower bounds, histories of property values and event sequences are necessary. Therefore, we need another extension which allows to represent the runtime history of a system. For this, we introduce a novel metamodeling blueprint which introduces the concept of history by providing a sequence of steps having a particular timestamp associated. Figure 5 illustrates the separation of steps into measurement snapshots and event snapshots. These specific steps are forming the event histories and measurement histories. The measurement history contains all measurement snapshots, which comprise values for given time steps. Event histories do the same for events. In our running example, the measurement snapshots refer to the value runtime concept introduced by the runtime extension and the event snapshots are referring to the transition firing concept.

Having this base structure introduced allows us to represent time-series data in design models by using runtime concepts as glue between models and data.

Runtime Properties
For analysing the time-series data represented in the aforementioned runtime history models, we introduce derived properties which actually represent runtime properties. Derived properties have been already used heavily in the past for deriving additional information from given structures and values. As we explicitly represent runtime histories as model structures, we can make use of derived properties to derive runtime information from the base time-series recorded during operation.

In the following we state three runtime properties for the given design language, namely for
the Property metaclass and the Assignment metaclass. We use standard OCL to derive the runtime properties.

For properties defined in blocks, it may be of interest if their values are strictly increasing over time or not. This can be expressed in OCL by providing a derived history reference for properties from the complete measurement history. The reference only contains the slice of the full history which concerns the given property. Using this reference, we can simply collect all values as a sequence (the ordering expresses the occurrence of the values). If the sorted sequence corresponds to the base sequence, then the property is strictly increasing.

```ocl
class Property
  // isStrictlyIncreasing: Boolean
  derived: self.history.steps.measure.value -> flatten() -> sortedBy(x) = self.history.steps.measure.value -> flatten()

end class Property
```

Concerning the assignments within states, one may be interested if the stated value is actually realized by the system. For this, the realized values may be collected by taking the last snapshots of all assignment executions for a given assignment.

```ocl
class Assignment
  // realizedValues: Set(Float)
  derived: self.histories -> collect(x)x.steps.last() -> collect(x)x.measure.value -> asSet()

end class Assignment
```

Having the set of realized values, the maximum deviation is calculated by introducing another derived property which builds on the previous one.

```ocl
class Assignment
  // maxDeviation: Float
  derived: self.realizedValues -> collect(x)|(self.value-x).abs() -> sortedBy(x) -> last()

end class Assignment
```

5 Related Work

In this section, we discuss existing work with respect to the contribution of this paper, namely the combination of model-driven and data-driven approaches with a focus on time-series analytics. Therefore, we first discuss data-driven approaches for enhancing existing domain specific languages (DSLs), and subsequently, we enumerate existing work which proposes dedicated DSLs for time-series analytics.

Data-driven approaches for DSLs

An emerging field for data-enhanced modelling languages is Web engineering. For instance, Bernaschina et al. (2017) point to the fact that there is the need for merging Web site navigation statistics of user behaviour with the structure of...
the Web application models. The authors show the advantages of combining user interaction models with user tracking information in form of user navigation logs, and details about the visualized content in the pages. Their approach interweaves design time information and runtime execution data of Web sites in order to significantly improve the analysis of user behaviour. In (Artner et al. 2017), we combined navigation models with Markov chains for representing navigation path probabilities, which are derived from execution logs. While these existing approaches for Web applications follow the general idea of combining data-driven and model-driven approaches, the approach of this paper is independent from the actual domain and may be also used in the future to reproduce these existing specific approaches.

Another very active research field is process mining (van der Aalst 2016) which aims to discover process models from workflow execution logs. A variety of process mining algorithms exists that allows the discovery of different process models in different formalisms. In (Wolny et al. 2017), we present an initial architecture how process mining may be related with time-series mining. By this, not only the dependencies between different process steps may be uncovered, but also dependencies between data and process steps are approachable.

Finally, in (Hartmann et al. 2017) the authors present a DSL which allows not only the specification of structural aspects of a systems, but also the definition of so-called learned properties. Such properties are computed from runtime data by using some kind of machine learning algorithms. Our approach directly allows to encode such properties as derived properties based on time-series data computed with OCL as we model the runtime history explicitly. In future work, it will be interesting to combine our time-series analysis with machine learning algorithms as proposed by Hartmann et al. (2017).

### DSLs for Time-Series Analytics

The OMS3 modelling framework introduces an extensible and lightweight layer for a simulation description expressed as Simulation DSL by using Groovy as a framework for providing the code generator implementation. In (David et al. 2012), the authors present DSL variants in OMS3, e.g., the Ensemble Streamflow Prediction (ESP) DSL. This DSL uses time-series of historic meteorological data as model input to predict future conditions. In their approach, DSLs are employed for time-series unlike in our approach, where we use time-series for domain-specific modelling.

Gekko is an open-source modelling approach for time-series data management and for solving and analysing large-scale time-series models. Gekko may be considered as a kind of DSL with a time-series domain focus. It provides interfaces to statistic packages such as R. In our approach, we use an open-source time-series database which offers besides high-availability storage and monitoring of time-series, application metrics and real-time analytics in addition. Nevertheless, in future work it is of interest to evaluate different possibilities to perform time-series analytics in addition to our current approach.

### 6 Conclusion and Future Work

In this paper, we have introduced an unifying architecture for combining model-driven and data-driven approaches for system engineering. By this architecture, we allow for specifying and computing runtime properties based on time-series data through design models. The extensions needed on the metamodel level are non-intrusive and connected to existing approaches for specifying the operational semantics of languages. The presented runtime history metamodel fragments are applicable for any design modelling language comprising features to be measured and events to be tracked as the current metamodeling languages Ecore and OCL are reused. We demonstrated our

---

6 https://alm.engr.colostate.edu/cb/project/oms
7 http://groovy-lang.org
8 http://t-t.dk/gekko
approach for a cyber-physical production system case. We have also realized a prototype in Eclipse supporting our approach which is available on our project website.\footnote{https://cdl-mint.big.tuwien.ac.at/case-study-artefacts-for-emisa-2017}

While the presented approach opens the door for using time-series analytics in a model-driven engineering toolbox, there are still several challenges to be tackled in the future. In particular, we consider the following points on our roadmap:

- \textit{scalability} (e.g., should the analysis be performed on the model level or directly in the time-series database?),
- \textit{expressivity} (e.g., which extensions of OCL are necessary for statistical reasoning?),
- \textit{understandability} (e.g., how to visualize time-series oriented information in diagrams?), and
- \textit{predictability} (e.g., how to derive and use operations from time-series for predicting future runtime states?).

References


10 Temporal Models on Time Series Databases

DOI: 10.5381/jot.2020.19.3.a14
ABSTRACT With the emergence of Cyber-Physical Systems (CPS), several sophisticated runtime monitoring solutions have been proposed in order to deal with extensive execution logs. One promising development in this respect is the integration of time series databases that support the storage of massive amounts of historical data as well as to provide fast query capabilities to reason about runtime properties of such CPS.

In this paper, we discuss how conceptual modeling can benefit from time series databases, and vice versa. In particular, we present how metamodels and their instances, i.e., models, can be partially mapped to time series databases. Thus, the traceability between design and simulation/runtime activities can be ensured by retrieving and accessing runtime information, i.e., time series data, in design models. On this basis, the contribution of this paper is four-fold. First, a dedicated profile for annotating design models for time series databases is presented. Second, a mapping for integrating the metamodeling framework EMF with InfluxDB is introduced as a technology backbone enabling two distinct mapping strategies for model information. Third, we demonstrate how continuous time series queries can be combined with the Object Constraint Language (OCL) for navigation through models, now enriched with derived runtime properties. Finally, we also present an initial evaluation of the different mapping strategies with respect to data storage and query performance. Our initial results show the efficiency of applying derived runtime properties as time series queries also for large model histories.

KEYWORDS Runtime Models, Query Languages, Model-Based Analysis, Temporal Modeling, Time Series Databases.

1. Introduction

With the emergence of Cyber-Physical Systems (CPS) and sophisticated runtime monitoring infrastructures, time series databases (Badel et al. 2017) are nowadays frequently applied to store historical data of systems as well as to provide powerful analysis by dedicated query languages.

At the same time, Model-Driven Engineering (MDE) (Brambilla et al. 2017) approaches are a promising line for dealing with the complexity of designing CPS. However, in recent years the scope of MDE has been also extended to runtime aspects of CPS (Mazak & Wimmer 2016; Benelallam et al. 2017; Cruz, Sadovykh, Truscan, Brunelière, et al. 2020; Bencomo et al. 2019; Gogolla et al. 2019; Kästner et al. 2018).

Several approaches for dealing with runtime data in models have been proposed which are often referred to temporal models in analogy to temporal databases (Gómez et al. 2018; Wolny et al. 2018; Bill et al. 2017). Temporal models go beyond representing and processing the current state of systems. By this, they extend research done in the last decades where several dedicated mappings from design models to different database technologies following different data paradigms have been proposed, e.g., see (Gogolla 2005). However, currently there is a lack of approaches which deal with the explicit mapping of design models to time series databases which can be considered as a special type of temporal databases (Schmidt et
Such mappings are required to further close the gap between design time modeling activities and simulation/runtime monitoring activities (Gogolla et al. 2019), which employ time series analytics. For instance, time series representations and analytics are foreseen in the development of SysML v2 (Wolny et al. 2020) in order to deal with additional activities in engineering technical systems such as computing activities in engineering technical systems such as computing

To tackle this limitation, we propose in this paper a novel partial mapping from metamodels and their instances, i.e., the models, to time series databases. The partial mapping deals with the fact that most often not all model elements contribute to a time series, and thus, only those elements which have a runtime history are explicitly mapped to time series structures. Therefore, we propose a dedicated profile for extending the metamodels with appropriate annotations to drive and optimize the generation of model-based time series database connectors.

In our approach, we focus on Ecore. For illustrating metamodeling, and time series databases.

2.1. Metamodeling

Model-driven Engineering (MDE) considers models as first class citizens (Bézivin et al. 2014). A model is used to describe an abstraction of reality for a specific purpose. The basis of such models are modeling languages which are defined by their metamodels. Metamodels are used to describe the abstract syntax of modeling languages. Models created by using a modeling language are instances of the metamodel, and thus, conform to it (Bézivin et al. 2014). One of the best known modeling languages (amongst others) is the Unified Modeling Language (UML) which bases on the Meta Object Facility (MOF) standard. The advantages of UML are platform independence as well as adaption and extension capabilities for users to meet their own requirements for a specific purpose. UML offers a wide range of views and different types of diagrams to represent the structure and behavior of a system to be modeled. One example of a metamodeling language which is based on a core subset of UML and MOF is Ecore from the Eclipse Modeling Framework (EMF). Since Ecore supports the key concepts of modeling as input to development and integration tools, it is one of the most widely used languages for code generation and model serialization for data interchange.

In our approach, we focus on Ecore. For illustrating metamodels and models we employ as concrete syntax UML class diagrams and UML object diagrams, respectively. Figure 1 shows excerpts of (a) Ecore’s concepts for defining metamodels and (b) Ecore’s concepts for representing instances of the metamodels, i.e., models. Models are represented by object graphs and consist of objects (instances of classes), slots

Figure 1 Excerpt of Ecore: (a) concepts for defining metamodels and (b) concepts for defining models.
for storing values (instances of attributes), calls for executing operations (instances of operations) with particular values (instances of parameters), and links between objects (instances of references).

UML object graphs have to conform to the given UML class diagrams. For instance, this means that if an object is existing in the object graph, a corresponding concrete class must exist in the metamodel which act as type for the object.

Additionally, with Ecore, metamodel elements might be annotated with further information (so-called annotations), e.g., for tagging elements for particular platforms or purposes as we notated with further information (so-called annotations).

On the basis of these two metamodels, Figure 2 shows an example of an excerpt of a Production System Domain Specific Language (PS DSL) and an example instantiation of it. The DSL in Figure 2(a) shows a system consists of various components. Each component has an unique id, a temperature value (temp), a property for showing if the component is active or not (isActive) and a method run() which is active when the system is busy processing an order.

Based on this discussed metamodel, Figure 2(b) shows an instance of a particular system. This system consists of three different components (c1-c3) with different property values. If c3 is starting to work (c3.run()), the property values of c3, more specific the temperature value (temp) and the isActive value, are changing. Thus, the system state is changing over time and in the shown example only snapshots of the system at a specific point in time are represented (Jogolla et al. 2014).

2.2. Time Series Database

A time series (TS) is a sequence of data points acquired by repeatedly measuring certain parameters (e.g., temperature) over time. The measured values are stored together with the timestamps at which the measurements are taken (Jensen et al. 2017). Although the measurements are usually performed at regular intervals (default in milliseconds), regularity is not a mandatory requirement. The increased interest on this data is in particular the result of the ongoing development in the CPS domain with its IoT technologies as described in the introduction, in which the number of sensors that regularly measure defined conditions is constantly increasing, e.g., for an efficient runtime monitoring.

Time series databases are used for storing, processing, querying as well as analyzing this data generated over time (Bader et al. 2017). Such data consists of timestamps, corresponding values, and optional tags which can consist of names and values (both mostly alphanumeric). Queries can be executed for timestamps or intervals without having to model the data into another structure (Bader et al. 2017). Since the TSDB is not only used for simply collecting data, the term “Time Series Database” (TSDB) is synonymous to the term “Time Series Database Management System” as a kind of software with specialized functions such as compressing or aggregating time series data (Kholod et al. 2017). As mentioned above such time series data is metering from a lot of different sensors. For storing these large amount of data with sufficiently high performance, TSDBs provide the relevant scalability (Jensen et al. 2019).

The level of granularity depends on the type of time series data and the requirements for data analysis, especially since not every time series has to be measured at the same level of detail in order to gain valuable insights of the monitored system (Bader et al. 2017). As an example, the half-hourly measurement of temperature in several rooms of an office building can be mentioned. In this example the granularity is 30 minutes. The values of a tag called “room” can then further specify to which room of the house the measured temperature (value of the time series) belongs.

Time series data differs from other data sets in that it is usually added as a new entry in a TSDB, and therefore, already stored entries are not overwritten (Kholod et al. 2017). Exceptions may only caused by the correction of faulty data, e.g., due to delayed measurements or a failure of sensors. Therefore TSDBs allow the recording and analysis of massive historical data, e.g., for anomaly detection or predictive analytics (Mazak et al. 2018). Thus, any changes over time can be traced nearly in a seamless manner. The storage of time series data, the analysis, and the monitoring of any changes over time provide a great deal of informative added value compared to other types of data, which can only represent a current status (Kholod et al. 2017). In our approach, we use InfluxDB\(^4\) an open source TSDB by which we can continuously store and query data independently of another DBMS (Bader et al. 2017). For querying, it provides a SQL-like language, and for storing it provides rules for (long-
3. Mapping Models to Time Series Representations

In order to allow an integration of time series storage and analysis in a model-based manner, in this section we present the design rationale for our approach before we outline two mapping strategies from object-oriented models (as described in Section 2) to TSDB.

3.1. A Polyglot for Combining Models with Time Series Databases

For combining models, especially EMF-based models, with TSDB, we aim for a polyglot solution where the static information resides in the model as it is already available, e.g., by term) data storage. For instance, InfluxDB enables flexible data aggregations based on the timing factor and running calculations of functions (e.g., average temperature per hour).

Figure 3 shows a metamodel of the TSDB. The TSDatabase has a specific name and consists of various Measurements. Each measurement must consist of a Timestamp and a FieldSet where the different time series values are stored. Optionally, the measurement can have some additional metadata stored in a TagSet. For instance, InfluxDB implements this metamodel and its line protocol informs the database of the measurement, tag set, field set, and timestamp. Listing 1 shows the structure of the line protocol, with first its measurement, followed by an optional TagSet, followed by a FieldSet with at least one field and optionally a timestamp. If no timestamp is specified, the current system time is taken by default.

Listing 1 Example of the line protocol of InfluxDB.

These two storage parts are combined by a common ModelAPI, which then can be accessed and used by various applications. This unifying API abstracts implementation details and allow for a similar way of working with models as it is provided by EMF out-of-the-box. In particular, we reuse as much as possible and only extend those parts which are really required. As a result, the model is applied as close as possible to the EMF standard and the required information for the TSDB can be attached in a light-weight manner. In order to achieve such unifying API with a polyglot there are various requirements that must be fulfilled. First of all, there has to be a built-in mechanism that determines which information from the model should be transferred to the TSDB and stored there. Second, there should be as well a procedure that extracts data from the TSDB by querying and displaying it back in the model. Third, our temporal extensions should not hinder or pollute the use of models and they should be still manipulated as before. The main goal is to embed this process into a conceptual schema to avoid hard coding the functionalities again and again for different cases. In the following subsections, we describe the design choices of the polyglot.
With the use of precision we use an OCL dialect that needs further processing before execution (cf. Figure 5). The profile defines different kinds of stereotypes for classes, structural features as well as operations. The stereotype Temporal indicates that these elements (classes or structural features such as attributes and references) are temporal features that should be recorded as time series in a TSDB.

The use of precision, the accuracy of the recordings can be defined from nanoseconds (ns) to seconds (s). The stereotype Tag should be used for features that represent important metadata of time series features. For instance, the id of the room where the temperature is measured. The stereotype DerivedRTProperty marks features that are derived runtime properties. Derived properties are features where the feature value is computed based on other feature values. Our stereotype is used for defining properties that get their values during runtime based on runtime data stored in the TSDB. For instance, the average temperature of a specific room over a whole day. We introduce this custom stereotype for our derived runtime properties since we use an OCL dialect that needs further processing before execution (cf. Section 3.4). Additionally, there are two stereotypes for operations: stereotype Log is for logging the start and the end of an operation and stereotype Reset is used to refresh the system state, e.g., for a new simulation run.

In Figure 6, we show an example application for the previously presented metamodel in Section 2. In particular, we show on the left hand side the usage of the profile to map on a fine grained level two properties as temporal while on the right hand side we configure the whole class as temporal. These two usages are reflecting the two mapping strategies which are explained next.

### 3.3. Mapping Strategies

Based on the above presented profile, we now establish the mapping between models and the TSDB. By this mapping, (i) the traceability between design and runtime activities should be enabled, (ii) and runtime information (i.e., time series data) should be retrieved and should be accessible through models. For this purpose, it must be decided how objects, slots, operation calls, and links from the models, i.e., object graphs (conform to classes, attributes, operations and references, respectively) are mapped to the TSDB elements such as measurements, tags, and fields.

In this paper, we consider two conceptual strategies for the M2TS mapper: (i) a strategy where it is possible to store each temporal property individually (cf. Section 3.3.1), and (ii), a strategy by which the whole object with all its associated information is stored (cf. Section 3.3.2). Of course, there exist various other combinations of strategies additionally to these two discussed ones. However, selecting a suitable strategy depends on performance, memory size, and general feasibility. In this paper, we focus on the presented ones, since they give already several configuration opportunities for modelers and consider the capabilities of the deployed TSDB functions. The developed TS profile is designed to cover both strategies, but is not limited to them. It can be extended as well as the mappings may be adapted to specific strategy changes.

#### 3.3.1. Single Property Mappings

The first strategy is to map single properties, e.g., the temperature of a room. The goal is to continuously log the progression of such property values in a TSDB and to query these values in terms of the models if necessary. This means that the property becomes a "temporal feature" with its own measurement. Only such time relevant data is stored in the TSDB. The remaining information, such as static metadata, is stored in the model, since such information is constant and does never change over time. However, to ensure that properties of different objects can be distinguished and that the relationship between them is not lost, the information to which object the temporal feature belongs must also be stored in the measurement. For example, if the average temperature of a specific room is required, it should be avoided that the average temperature of all rooms is analyzed. Therefore, the room id is important to be related to the measurement.

The table in Appendix A shows the mapping of the object diagram elements to the specific elements of the TSDB based on the TS profile. It has to be mentioned that tags are optional additional information and fields are mandatory for value recording. Similarly, each measurement in the TSDB contains a mandatory timestamp, where precision can be used to determine the accuracy. For DerivedRTProperties a query is executed on the TSDB which returns a series as a result (cf. Section 3.4).

```
10 Composite Operation Modeling By-Example

Temporal Models on TSDBs

3.2. Time Series Profile

In a first step, we propose a profile (realized with EMF Annotations) for extending existing metamodels by time series aspects (cf. Figure 5). The profile defines different kinds of stereotypes for classes, StructuralFeatures as well as Operations. The stereotype Temporal indicates that these elements (classes or structural features such as attributes and references) are temporal features that should be recorded as time series in a TSDB.

With the use of precision, the accuracy of the recordings can be defined from nanoseconds (ns) to seconds (s). The stereotype Tag should be used for features that represent important metadata of time series features. For instance, the id of the room where the temperature is measured. The stereotype DerivedRTProperty marks features as derived runtime properties. Derived properties are features where the feature value is computed based on other feature values. Our stereotype is used for defining properties that get their values during runtime based on runtime data stored in the TSDB. For instance, the average temperature of a specific room over a whole day. We introduce this custom stereotype for our derived runtime properties since we use an OCL dialect that need further processing before execution (cf. Section 3.4). Additionally, there are two stereotypes for operations: stereotype Log is for logging the start and the end of an operation and stereotype Reset is used to refresh the system state, e.g., for a new simulation run.

In Figure 6, we show an example application for the previously presented metamodel in Section 2. In particular, we show on the left hand side the usage of the profile to map on a fine grained level two properties as temporal while on the right hand side we configure the whole class as temporal. These two usages are reflecting the two mapping strategies which are explained next.

```
Additional to this example, annotated tags will be added to temporal features of the same object.

Based on the example of Figure 2(b), the following listings show the entries of the different measurements in the database. Listing 3 shows the stored values of the three different components c1-c3, before executing c3.run()). For the two temporal properties inActive and temp, the initial values are stored with their timestamp.

Listing 3 TSDB entries for the single property strategy before c3.run() is executed.

```
c3.run()
```

```
measurement obj502838712:
  time: 158940717116594100
  value: 1443055846
  inActive: false
  run: 502838712
```

Listing 4 TSDB entries for the single property strategy after c3.run() is executed.

```
c3.run()
```

```
measurement obj502838712:
  time: 1589407171200396800
  value: 1443055846
  inActive: true
```

The table in Appendix B gives an overview of the complete object mapping strategy, how the individual annotated elements from the model are stored in the TSDB. Based on this, Listing 5 shows a pseudo code line protocol template, again embedded in an ECA rule, for storing complete objects as measurements and their features as fields. In addition, if structural features are annotated as tags, then they would be saved as tags in the object measurement.

```
ON
  object.set(feature, value)
  IF
    object.class is Temporal
THEN
time = UnixTimestamp(precision = <<object.class.
temporal_precision>> // default: ns
db.insert(measure<<object.id>> FOR EACH f in
  <<object.features>>|fieldN=[of name],
  <of.value>]) time)
```

Listing 5 Template of complete object mapping.

In comparison to the presented single property mapping (cf. Section 3.3.1), Listing 6 and Listing 7 show the set-up of the database and its entries for the complete object mapping. The structure of the information has changed and therefore also the structure of the TSDB queries (cf. Section 3.4) depends on the corresponding mapping.

```
c3.run()
```

```
measurement obj40170008:
  time: 158940689716594100
  value: 1443055846
  inActive: false
```

Listing 6 TSDB entries for the complete object strategy before c3.run() is executed.

```
c3.run()
```

```
measurement obj40170008:
  time: 1589406897200396800
  value: 1443055846
  inActive: true
```

3.3.2. Complete Object Mappings

The second strategy does not map single properties of objects in isolation but rather the entire object at once. This means that individual properties do not have to be annotated as temporal features, but the containing classes, and thus, the associated objects with their properties are stored in the database as measurements.
3.4. Query Capabilities

On the basis of the TS profile and the applied mapping strategies, we now present the query capabilities of our approach. In a first step, we present four basic operations for temporal properties, the first two are adapted from previous work (Gómez et al. 2018), and the last two are extensions:

1. getValueAt(Instant t)
   - Result: DataType value
   - Example code snippet:
   ```java
   TSQuery = SELECT self.temp FROM <<object.id>> WHERE time = t
   ```

2. getValueBetween(Instant t1, Instant t2)
   - Result: Map<Instant time, DataType value>
   - Example code snippet:
   ```java
   TSQuery = SELECT self.temp FROM <<object.id>> WHERE time BETWEEN t1 AND t2
   ```

3. getEndTimePointsForValue(Instant t)
   - Result: List<Instant time>
   - Example code snippet:
   ```java
   TSQuery = SELECT time FROM <<object.id>> WHERE self.temp = value1 AND time = t
   ```

4. getEndTimePointsForValueBetween(DataType value1, DataType value2)
   - Result: Map<Instant time, Boolean>
   - Example code snippet:
   ```java
   TSQuery = SELECT time FROM <<object.id>> WHERE self.temp BETWEEN value1 AND value2
   ```

Based on the used mapping strategy, the query implementation in the background, i.e., in the TSDB, differs, since there is a different data structure used in the TSDB. For instance, the following Listing 8 shows the difference for the basic operation (1).

Listing 8: TSDB query for getValueAt(Instant t) based on simple property mapping and complete object mapping.

**Single property mapping**:
```java
getValueAt(Instant t)
```

**Complete object mapping**:
```java
Object mapping:
```
```java
DerivedRTProperty(query:TS-OCL)
```

```
Listing 9: Pseudo code for the calculation of the utilization time of a component.
```
```
1589407171206409400 true 50
1589407171190348700 obj502838712 start
```

Additionally to these four basic operations, derived properties can be annotated with DerivedRTProperty(query:TS-OCL). As a first realization, the TS-OCL query must be expressed in the syntax of the query language of the TSDB (in the InfluxDB case it is InfluxQL), or in combination with standard OCL for navigation through the model (i.e., using self, navigation operators, etc.) The combination of OCL with InfluxQL is performed as a pre-processor approach. OCL is used to query the model elements which are injected into the InfluxQL query. The M2TS mapper is resolving the model elements to database entries and thus completes the InfluxQL query.

As an example, we consider as a derived property the maximum temperature of a component. Listing 10 shows the TS-OCL query for this example and the respective conversion to a TSQuery based on the two different mapping strategies.

Listing 10: Example query code of a derived runtime property.

```
DerivedRTProperty: MaxTemperature
```

```
Listing 10 Example query code of a derived runtime property.
```

These query capabilities enable the M2TS mapper not only to inject data to the TSDB from model changes, but also to extract data from the TSDB by model-based queries. As the derived runtime properties are in essence standard derived properties, they can be simply reused in standard OCL queries. Finally, the combination of OCL with InfluxQL allows to write model-based queries without having to deal with the concrete mapping approach in use.

4. Evaluation

In this section, we present and discuss the performance and scalability of our approach using a case study based on the PS-DSL metamodel (cf. Section 2, Figure 2 (a)). From a methodological view, we follow the guidelines for conducting case studies by Runeson and Höst (Runeson & Höst 2009) for performing the evaluation. The implementation of our approach and evaluation results can be found at our project website6.

4.1. Research Questions

Our general evaluation interest is the comparison of the two presented mapping strategies for our M2TS mapper on basis of performance and scalability. Therefore, we aim to answer the following research questions (RQs):

RQ1—Scalability of the database size with single property mapping vs. complete object mapping: How does the database size develop regarding different number of model changes and number of entries? Does the database size grow linear to model changes?

RQ2—Performance of the database: How does the database perform regarding different number of model changes and number of entries? Does the database perform consistently or does its performance vary?

RQ3—Scalability of the database size with single property mapping: How does the database size develop regarding different number of model changes and number of entries? Does the database size grow linear to model changes?

RQ4—Performance of the database: How does the database perform regarding different number of model changes and number of entries? Does the database perform consistently or does its performance vary?

5 https://www.omg.org/spec/OCL
6 https://cdl-mint.se.jku.at/case-study-artefacts-jot-2020/
changes? Is there a significant difference between the two mapping strategies?

**RQ2—Performance of the runtime queries for single property mapping vs. complete object mapping:** How long do the queries take for (i) values at a given timestamp, (ii) timestamps for specific values, and (iii) aggregate calculations such as average, maximum, and modal values? Is there a significant difference observable for the two mapping strategies?

### 4.2. Case Study Design

**Requirements:** As an appropriate input for our case study, we first require a system based on an Ecore model, which is annotated by our TS profile. The corresponding models must be executable and contribute to time series when executed. In addition, we require InfluxDB as running TSDB to store value records.

**Setup:** For our evaluation, as already mentioned, we use instances based on the PS-DSL metamodel. Our execution system consists of different numbers of components and the run method of each component is executed for various numbers of time.

Table 1 gives an overview of the different evaluation settings regarding number of components, number of runs, and number of entries in the TSDB. For instance, one setting consists of 100 components, 100 runs are executed for each component, and finally 80000 entries are stored in the TSDB. During simulation, the values of the properties `isActive` and `temp` are changing over time and logged in the TSDB based on the respective mapping strategy.

<table>
<thead>
<tr>
<th>No.</th>
<th>Set 1</th>
<th>Set 2</th>
<th>Set 3</th>
<th>Set 4</th>
<th>Set 5</th>
<th>Set 6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Comp.</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>Run()</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>Entries</td>
<td>80 K</td>
<td>8 M</td>
<td>20 M</td>
<td>32 M</td>
<td>72 M</td>
<td>128 M</td>
</tr>
<tr>
<td></td>
<td>200 M</td>
<td>500 M</td>
<td>200 M</td>
<td>500 M</td>
<td>200 M</td>
<td>500 M</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Number of entries</th>
<th>single property mapping</th>
<th>complete object mapping</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>500</td>
<td>200</td>
</tr>
<tr>
<td>100</td>
<td>500</td>
<td>200</td>
</tr>
<tr>
<td>100</td>
<td>500</td>
<td>200</td>
</tr>
<tr>
<td>100</td>
<td>500</td>
<td>200</td>
</tr>
<tr>
<td>100</td>
<td>500</td>
<td>200</td>
</tr>
<tr>
<td>100</td>
<td>500</td>
<td>200</td>
</tr>
</tbody>
</table>

Table 1 Number of (i) components in the model, (ii) run() executions for different settings, and (iii) entries in the TSDB.

For these settings, the query performance is evaluated as follows. On the one hand, for the different derived runtime properties, i.e., the maximum, mean, and mode values of the `temp` attribute for a selected component is calculated, and on the other hand, the general methods provided by our approach `getTimePoints`, `getValueAt` and `getValueAt()` are executed for particular values and time points.

For answering our RQs, we calculate the different durations for storing data, and for each query by `System.nanoTime()` in Java based on nanoseconds (ns). The performance is measured on an Acer Aspire VN7-791 with an Intel(R) Core(TM) i7-4720 HQ CPU @ 2.60 GHz, 2.60 GHz, with 16 GB of physical memory, and running Windows 8.1. 64 bits operating system.

Please note that we measured the CPU time by executing each mapping five times for all different settings and calculated the arithmetic mean of these runs. We use EMF, JDK 13 (important for precision accuracy of nanoseconds), and InfluxDB 1.8.0 to execute our approach.

**Prototype:** In a first prototypical implementation, we realized our M2TS mapper for EMF. In particular, we provide annotations for the metamodeling language Ecore with respect to utilizing the TSDB InfluxDB. The different stereotypes of our TS profile are implemented as EAnnotations on the Ecore model. For connecting the database, we make use of the open source Java client for InfluxDB and provide our own InfluxDBConnector which provides the glue between EMF models and InfluxDB. For automation purposes, we adapt the existing Java Emitter Templates (JET) for the EMF code generation. Thus, by the extended code generator we are able to provide an enriched API for EMF models to deal with temporal information, i.e., storage and query capabilities.

### 4.3. Results

In this subsection, we present the measurements for answering our research questions.

![Figure 7 Database size in relation to number of entries for both mapping strategies (in MB).](https://via.placeholder.com/150)

**Answering RQ1—Scalability of database sizes:** Our investigations regarding the TSDB size on the basis of the model changes show that both strategies show a linear increase (cf. Figure 7). It can be recognized that the size of the database for complete object mapping strategy increases slightly faster than for single property mapping strategy. However, this can be explained by the fact that whenever a value of a property is changed, the entire object is stored with a new timestamp.

**Answering RQ2—Performance of runtime queries:** Figure 8 shows the measurements of the query duration for both mapping strategies. In general, the queries are fast, as they take only from 1ms to about 7ms, depending on the entries in the TSDB. However, as the size of the database increases, the queries for `MeanMaxMode` and `getValueAt` for the single property mapping become slightly slower than in the case of the complete object mapping. This can be explained by the fact that starting from a certain number of entries, it plays a role whether the possible results have to be selected first (for single mapping using `object.id`), or are already selected and only need to be screened (for the complete mapping strategy, the object has its own measurement). However, based on a hypothesis testing (i.e., Wilcoxon rank-sum testing (Venables & Ripley 2002)), there is no significance regarding the difference between the

---

5 https://github.com/influxdata/influxdb-java
two mapping strategies: $p$-value $= 0.4848$, $H_0$: single property $=$ complete object; $p$-value $> 0.05$. Therefore, $H_0$ is not rejected.

4.4. Critical Discussion

In summary, the introduced mapping strategies both have advantages as well as limitations. The right choice of strategy mainly depends on the task to be accomplished, i.e., which queries are subsequently evaluated. Imagine you aim to query all instances for a given type. This would be straightforward for the complete object mapping strategy. Imagine you would like to query the max temperature for all components. This would be faster for the single property mapping.

Overall, the evaluation demonstrated the feasibility of both strategies concerning the data storage and query performance. However, we cannot generalize our results beyond our initial case study. First, we have to mention that there may be other cases where larger objects, i.e., objects having many slot values and links, have to be stored. Consequently, a higher size of the databases may be expected, and this may call for new strategies of mapping objects with only a partial subset of their slots and links. In future studies, we plan to evaluate settings in a larger context such as building a monitoring systems for an IoT network or building a runtime-based verification tool as well. Such studies will allow a more practice-oriented evaluation of the different strategies which may be further collected in a particular benchmark for temporal models.

Another feature to exploit from the TSDB may be the down sampling capability for data for given time frames, i.e., aggregating data from milliseconds to sec to minute ranges and so on. Moreover, the explicit usage of tags instead of fields has to be evaluated in future work. Finally, we would like to mention that our presented case study with all the corresponding artefacts is provided online and may be used by the research community as experimental test bed for future studies concerning finding appropriate mappings from models to time series databases.

5. Related Work

With respect to the contribution of this paper, namely, the mapping and connection of conceptual models to TSDBs, we discuss various threads of related work. First, we discuss temporal modeling approaches for validation and verification of models. Second, we present approaches for linking design and runtime models. Third, we explore approaches for versioning of models in temporal repositories. Finally, we discuss approaches that combine modeling languages with time series analytics.

5.1. Temporal Modeling Languages

There is abundant research on temporal extensions for modeling languages to specify the temporal characteristics of the system data (e.g., consider (Gregersen & Jensen 1999) for a survey), but not regarding the temporal dimensions of models themselves.

Further works advance these first attempts by extending also the query languages with temporal properties, mainly to enable the validation and verification of temporal properties on the data. Temporal OCL (TOCL) (Ziemann & Gogolla 2003) and Temporal UML (Cabot et al. 2003) are two examples of OCL extensions for the evaluation of temporal constraints.

Temporal extensions have also been applied to specific types of systems (e.g., adaptive systems (Mouline et al. 2018)) and DSLs (e.g. timed Petri nets (Bender et al. 2008)). Even TOCL, which can be seen as a generic language, can also be used as an component in other DSLs as described in (Meyers et al. 2014).

In this line, (Bousse et al. 2019) discuss and apply a pattern to extend modeling languages with events, traces, and further runtime concepts to represent the state of a model’s execution and to use TOCL for defining properties that are verified by mapping the models as well as the properties expressed in TOCL.
to formal domains that provide verification support. Efficiency of these types of temporal inspection queries is also the focus of (García-Dominguez et al. 2018) and (García-Dominguez et al. 2019).

Nevertheless, all these approaches (including our own previous TemporalEMF proposal (Gómez et al. 2018)) are mostly oriented towards the retrieval of specific past states of the model/data, elaborating on the concepts of valid time and transaction time of (bi)temporal models. Instead, in this work we explicitly focus on the support for complete time series storage and analysis, which opens the door to more powerful and rich possibilities, like the computation of different KPIs for models as part of design exploration and simulation scenarios.

5.2. Linking Design-time and Runtime Models

In this subsection, we discuss approaches using traceability between design and runtime models. The evolutionary aspect of engineering artifacts refers to the fact that they change over time. Models in engineering processes, e.g., usually develop from initial ideas to first drafts. They are then continuously revised, often by taking into account feedback from other resources, until they are finally released. However, also the feedback after the release from the operation should be reflected in those models to make traceability between design and operation feasible (Mazak & Wimmer 2016).

For this purpose, the authors of (Wolny et al. 2018) present an architecture to map runtime data back to the model level by using standard metamodeling techniques. Thereby, they do not only develop a unifying architecture for creating model snapshots on-the-fly, but to map the history of operation concerning certain properties. This allows to specify and compute runtime properties based on time series data through design models. This means, design-oriented languages are equipped with extensions for representing runtime states as well as runtime histories, which in turn allow the formulation and computation of runtime properties with OCL. This makes it feasible to directly interpret measurements within design models without introducing an impedance mismatch. The challenge with using OCL for this purpose is that even simple mathematical calculations (e.g., computing upper bounds or averages) may quickly become complex with respect to their definition and evaluation. For better scalability such calculations should be directly performed in the TSDB as we allow by the presented work of this paper.

If the design model is not yet coupled with its runtime counterpart, i.e., no annotations are made at model level, the authors of (Wolny et al. 2019) present an approach to transform raw sensor log data to UML sequence diagrams for graphical representation. Therefore, they provide a text-to-model transformation to transform text-based traces of a running system to UML sequence diagrams. As a basis for reconstructing such UML sequence diagrams, they develop a metamodel for representing system logs in an object-oriented manner. This makes it feasible to express system logs explicitly as models. However, they only use the time aspect to trace the correct order of the performed operations, but not to store the execution time, e.g., to be able to annotate information about average duration. This could be complemented by the approach presented in this paper.

Another project that also deals with the connection of design and runtime is the project MegaM@RT2\(^1\). In this scalable model-based framework for continuous development and runtime validation of complex systems trace links between design models and runtime are established based on bidirectional transformations (Cruz, Sadový, Truscan, Bruneliere, et al. 2020). Temporal aspects as we discuss in the context of this paper are not explicitly considered. However, the MegaM@RT2 approach is applicable for already existing systems which may be combined with our approach to enrich existing systems with TS collection and analysis.

5.3. Temporal Model Repositories

In (Bill et al. 2017), the authors discuss the need for temporal model repositories and the explicit representation of time in models. They discuss the gap of traditional Version Control Systems (VCS) such as SVN and Git, where each version of an evolving model is stored with a timestamp for the whole model (Altmanninger et al. 2009). While versioning the whole model is suitable for many development tasks, it makes it challenging to trace the evolution of specific model elements over time. Furthermore, the authors discuss several challenges when moving towards temporal model repositories such as (i) model storage, (ii) model access, (iii) model consistency, (iv) model manipulation, and (v) model visualization. In this paper, we have mostly focused on the first two points.

In the work presented in (Hartmann et al. 2014), the authors present an approach for versioning on the model level, i.e., Models@runtime to handle the history of data. They state that especially for the Models@run.time paradigm (Blair et al. 2009), which propagates the use of models to support runtime reasoning, an efficient mechanism is needed to store and navigate the history of model element values. Therefore, model elements have to be versioned independently from each other. Furthermore, they simplify and improve the performance of navigating between model elements coming from different versions by defining a navigation context for navigating in two dimensions (space and version). However, the versions have to be explicitly introduced and managed as in the aforementioned versioning systems. In our approach, we store individual model element or even individual properties with their associated timing aspects.

To tackle the discussed challenges in (Bill et al. 2017), in (Gómez et al. 2018) we present a temporal model infrastructure built on top of EMF—TemporalEMF. In summary, we showed how TemporalEMF enables to treat conceptual schemas as temporal models. On these models, temporal queries can be performed to retrieve model contents at different time points, e.g., to compare model content and to trace model states in the past. The TemporalEMF approach bases on concepts from temporal languages. The history of a model is transparently stored in a NoSQL database (i.e., HBase\(^8\)). In our newly presented approach no dependence to other DBMS, such as NoSQL ones, is needed, since we use a TSDB to reason about the history of property values accessible in the model.

---

1. https://megamart2-ecsel.eu/
3. \(^8\)
In (Haeseler et al. 2019), the authors discuss the need for tool support in the area of IT Landscape documentation. Therefore, they present a solution for storing, versioning, and querying of such IT Landscape models by means of an open source graph-based EMF model repository. In addition, the modular architecture allows to consider those models still as standalone components outside the repository context. A limitation is that ChronoSphere operates in local deployments, and therefore, is currently not distributed across several machines for greater scalability. In our approach, models and the TSDB can run separately on different machines. Since, our polyglot approach provides a ModelAPI, it could be considered in the ChronoSphere repository as well, which is generic, and therefore, not limited to the domain of IT Landscape documentation. This allows other applications to use the provided functionalities of our API for various use cases.

5.4. Modeling Languages for Time Series Analytics

In (David et al. 2012), the authors present the OMS310 modeling framework which provides an extensible and lightweight layer for simulation description expressed as so-called “Simulation DSL” based on Groovy11. The authors propagate DSLs for completing General Purpose Languages (GPLs) for specific simulation purposes. In their work, they present DSL variants in OMS3 such as a DSL for Ensemble Streamflow Prediction (ESP) based on meteorological time series data for predicting future conditions. Instead of creating a DSL for a specific purpose, in our approach, we propose a dedicated profile for extending metamodels with appropriate annotations to exist existing metamodels (e.g., of GPLs) by time series aspects.

Gekko12 is an open source modeling approach for time series data management and for solving, as well as analyzing large-scale time series models. It could be considered as a kind of DSL with a strong time series domain focus. It provides interfaces to statistical computing and graphics packages such as R13. In our approach, we use InfluxDB which offers besides high-availability storage and monitoring of time series data, application metrics as well as real-time analytics.

In this context, we also mention TimescaleDB14 which is an extension of PostgreSQL15. TimescaleDB is specially optimized for time series data in order to automatically partition data by time. Like PostgreSQL, TimescaleDB stores the data in a RDBMS and supports SQL as query language. Furthermore, it provides additional features for analyzing and manipulating time series data. Similar to the InfluxDB, TimescaleDB offers the possibility of a continuous calculation of functions. In particular such functions are queries that are executed continuously and in real time on the incoming data. The results of these regular queries are also stored in the TSDB as specified metrics (e.g., average room temperature every half hour with the applied metric). External tools such as Grafana16 or Tableau17 may also be used to visualize and analyze time series data. In addition to Grafana, the open source statistics software R for analyzing time series data should also be mentioned. However, the probably most extensive functionalities for querying data, setting warnings, and visualizing time series data is offered by InfluxDB, respectively by the InfluxData platform. Moreover, long-term storage of data is only provided by InfluxDB, and only to a limited extent by TimescaleDB. Additionally, the data scripting and query language Flux18 can be used in combination with InfluxDB. This standalone tool is optimized, e.g., for monitoring and provides built-in functions as well as importable packages to retrieve, transform, process, and output time series data. In contrast, our approach looks at TSDBs from a model-driven perspective and how conceptual modeling and TSDBs can benefit from each other.

6. Conclusion and Future Work

In this paper, we have presented a novel set of partial mappings from models to TSDBs. In particular, we presented a profile to annotate metamodels in order to automatically generate wrappers to time series databases that enable storing model updates as well as querying historical model information. Two different mapping strategies are proposed and evaluated in terms of their feasibility and scalability. While the current work presents interesting insights how modeling technologies may be combined with TSDB, we foresee several additional lines of research worth to investigate in addition to the ones mentioned in the evaluation section.

On the modeling side, we need to deal with co-evolution issues given that the TSDB is schema-less. For usability reasons, we would also like to be able to express complex time-related queries in OCL (e.g., by pre-defining a set of time-series operators, similar to what we did in (Cabot et al. 2010) for multidimensional models).

On the mapping side, we will investigate how to run approximate queries to deal with a variety of uncertainty scenarios (Burguello et al. 2019) and study the potential of combining both temporal and time-series information. This would enable even more complex analysis where we could, for instance, evaluate whether a new design model behaves better than one we used in the past by comparing their respective associated time-series data. It even allows to forecast the expected behavior of future designs. Finally, we are interested in mapping and storing not only the models themselves but also all modeling operations on them (e.g., by storing the trace information automatically created by some transformation engines such as ATL).

Acknowledgments

The work has been supported by the Austrian Federal Ministry for Digital and Economic Affairs and the National Foundation for Research, Technology and Development (CDG) and by the Austrian Federal Ministry for Education, Science and Research.

10 https://didt.engr.colostate.edu/DSS/wiki/1661
11 https://groovy-lang.org
12 http://t.t.dk/gekko
13 https://www.j-project.org
14 https://www.timescale.com
15 https://www.postgresql.org
16 https://grafana.com
17 https://www.tableau.com
18 https://www.influxdata.com/products/flux
in the TransIT Project (BMWFW-11.102/0033-IV/8/2019) as well as by the FWF under the Grant Numbers P28519-N31 and P30525-N31, and from the Spanish government under project Open Data for All (RETOS TIN2016-75944-R).

References


A. Single Property to TSDB Mapping Table

<table>
<thead>
<tr>
<th>Model</th>
<th>TS Profile</th>
<th>measurement</th>
<th>tag key</th>
<th>tag value</th>
<th>T</th>
</tr>
</thead>
<tbody>
<tr>
<td>Slot</td>
<td>Temporal (precision: TimeUnit)</td>
<td>attribute name</td>
<td>“object”</td>
<td>object.id</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>-</td>
<td>attribute</td>
<td>slot.value</td>
<td></td>
</tr>
</tbody>
</table>

| Derived Slot | DerivedRTProperty (query: TS-OCL) | - | - | - | - |

| Link | Temporal (precision: TimeUnit) | reference name | “object” | object.id | re | eq | m |
|      |                                | -             | -         | -         |    |    |   |

| Derived Link | DerivedRTProperty (query: TS-OCL) | - | - | - | - |

<table>
<thead>
<tr>
<th>Method</th>
<th>Log</th>
<th>method.name</th>
<th>“object”</th>
<th>object.id</th>
<th>“s”</th>
</tr>
</thead>
</table>

B. Complete Object to TSDB Mapping Table

<table>
<thead>
<tr>
<th>Model</th>
<th>TS Profile</th>
<th>measurement</th>
<th>tag key</th>
<th>tag value</th>
<th>T</th>
</tr>
</thead>
<tbody>
<tr>
<td>Object</td>
<td>Temporal (precision: TimeUnit)</td>
<td>object.id</td>
<td>-</td>
<td>-</td>
<td>for attr for ref</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Slot</th>
<th>Tag</th>
<th>attribute name</th>
<th>slot.value</th>
</tr>
</thead>
</table>

| Derived Slot | DerivedRTProperty (query: TS-OCL) | - | - | - | - |

<table>
<thead>
<tr>
<th>Link</th>
<th>Tag</th>
<th>reference name</th>
<th>target.id</th>
</tr>
</thead>
</table>

| Derived Link | DerivedRTProperty (query: TS-OCL) | - | - | - | - |

<table>
<thead>
<tr>
<th>Method</th>
<th>Log</th>
<th>method.name</th>
<th>“object”</th>
<th>object.id</th>
<th>“s”</th>
</tr>
</thead>
</table>
About the authors

Alexandra Mazak is a Professor at Montanuniversität Leoben in the research unit Subsurface Engineering. Before she headed Module 3 in the Christian Doppler Laboratory for Model-Integrated Smart Production (CDL-MINT) at JKU Linz. Her research field focuses on digital transformation in tunnel information modeling as well as in modeling of production systems. Her research interests are on information integration, model-driven engineering and model-based data analytics. She headed numerous interdisciplinary national funded projects in the field of digital transformation. You can contact the author at alexandra.mazak-huemer@unileoben.ac.at or visit https://www.tunnellinghub.at/.

Sabine Wolny is currently working as PhD student at the JKU Linz in the module Reactive Model Repositories of the CDL-MINT and in the Research Unit of Building Physics at TU Wien with focus on project management and software development. Her research interests include SysML-based modeling, execution/simulation of production systems, reverse engineering, data integration and back-propagation of runtime information. You can contact the author at sabine.wolny@jku.at or visit https://www.se.jku.at/sabine-wolny/.

Abel Gómez is a senior researcher of the Internet Interdisciplinary Institute, and an Assistant Professor at the Faculty of Computer Science, Multimedia and Telecommunications, both belonging to the Universitat Oberta de Catalunya in Spain. His research interests fall in the broad field of model-driven engineering (MDE), and his research lines have evolved in two complementary directions: the development of core technologies to support MDE activities, and the application of MDE techniques to solve Software Engineering problems. You can contact the author at agomezlla@uoc.edu or visit https://abel.gomez.llana.me.

Jordi Cabot is currently an ICREA Research Professor with the Internet Interdisciplinary Institute. His research interests include software and systems modeling, formal verification, and the role of AI can play in software development (and vice versa). He has published over 200 peer-reviewed conference and journal articles on these topics. Apart from his scientific publications, he writes and blogs about all these topics in several sites. You can contact the author at jordi.cabot@icrea.cat or visit https://jordicabot.com/.

Manuel Wimmer is Full Professor leading the Institute of Business Informatics - Software Engineering at the Johannes Kepler University Linz and he is the head of the Christian Doppler Laboratory CDL-MINT. His research interests comprise foundations of model engineering techniques as well as their application in domains such as tool interoperability, legacy modeling tool modernization, model versioning and evolution, and industrial engineering. You can contact the author at manuel.wimmer@jku.at or visit https://www.se.jku.at/manuel-wimmer/.

Gerti Kappel is Full Professor at the Institute of Information Systems Engineering at TU Wien, chairing the Business Informatics Group. Since the beginning of 2020 she acts as the dean of the Faculty of Informatics at TU Wien. Her current research interests include Model Engineering, Web Engineering, and Process Engineering, with a special emphasis on cyber-physical production systems. You can contact the author at kappel@big.tuwien.ac.at or visit https://www.big.tuwien.ac.at/people/gkappel/.
11 Execution-Based Model Profiling

A. Mazak, P. Patsuk-Bösch and M. Wimmer;
Data-Driven Process Discovery and Analysis- 6th IFIP WG 2.6 International Symposium (SIMPDA), Revised Selected Papers, Vol. 307, LNBIP, Springer, (2016), pp. 37–52. DOI: 10.1007/978-3-319-74161-1_3
Execution-Based Model Profiling

Alexandra Mazak(B), Manuel Wimmer, and Polina Patsuk-Bösch

Christian Doppler Laboratory for Model-Integrated Smart Production (CDL-MINT),
Institute of Software Technology and Interactive Systems, TU Wien,
Favoritenstrasse 9-11, 1040 Vienna, Austria
{mazak,wimmer,patsuk}@big.tuwien.ac.at
https://cdl-mint.big.tuwien.ac.at

Abstract. In model-driven engineering (MDE), models are mostly used in prescriptive ways for system engineering. While prescriptive models are indeed an important ingredient to realize a system, for later phases in the systems’ lifecycles additional model types are beneficial to use. Unfortunately, current MDE approaches mostly neglect the information upstream in terms of descriptive models from operations to (re)design phases. To tackle this limitation, we propose execution-based model profiling as a continuous process to improve prescriptive models at design-time through runtime information. This approach incorporates knowledge in terms of model profiles from execution logs of the running system. To accomplish this, we combine techniques of process mining with runtime models of MDE. In the course of a case study, we make use of a traffic light system example to demonstrate the feasibility and benefits of the introduced execution-based model profiling approach.

1 Introduction

In model-driven engineering (MDE), models are put in the center and used as a driver throughout the software development process, finally leading to an automated generation of the software systems [14]. In the current state-of-practice in MDE [3], models are used as an abstraction and generalization of a system to be developed. By definition, a model never describes reality in its entirety, rather it describes a scope of reality for a certain purpose in a given context [3]. Thus, models are used as prescriptive models for creating a software system [11]. Such models@design.time determine the scope and details of a domain of interest to be studied. Thereby, different aspects of the domain or of its solution can be taken into account. For this purpose different types of modeling languages (e.g., state charts, class diagrams, etc.) may be used. It has to be emphasized that engineers typically have the desirable behavior in mind when creating a system, since they are not aware in these early phases of the many deviations that may take place at runtime [23].

According to Brambilla et al. [3] the implementation phase deals with the mapping of prescriptive models to some executable systems and consists of three levels: (i) the modeling level where the models are defined, (ii) the realization
level where the solutions are implemented through artifacts that are used in the running system, and (iii) the automation level where mappings from the modeling to the realization phase are made. Thus, the flow is from models down to the running realization through model transformations.

While prescriptive or design models are indeed a very important ingredient to realize a system, for later phases in the system’s lifecycle additional model types are needed. Therefore, descriptive models may be employed to better understand how the system is actually realized and how it is operating in a certain environment. Compared to prescriptive models, these other mentioned types of models are only marginal explored in the field of MDE, and if used at all, they are built manually. Unfortunately, MDE approaches have mostly neglected the possibility to describe an existing and operating system which may act as feedback for improving design models. As theoretically outlined in [16], we propose model profiling as a continuous process (i) to improve the quality of design models through runtime information by incorporating knowledge in form of profiled metadata from the system’s operation, (ii) to deal with the evolution of these models, and (iii) to better anticipate the unforeseen. However, our aim is not to “re-invent the wheel” when we aim to close the loop between downstream information derived from prescriptive models and upstream information in terms of descriptive models. There exist already promising techniques to focus on runtime phenomena, especially in the research field of Process Mining (PM) [23]. Thus, our model profiling approach in its first version follows the main idea of combining MDE and PM. The contribution of this paper is to present a unifying architecture for a combined but loosely-coupled usage of MDE approaches and PM techniques.

The remainder of this paper is structured as follows. In the next section, we present a unified conceptual architecture for combining MDE with PM frameworks. In Sect. 3, we present a case study of execution-based model profiling conducted on a traffic light system example and present the results. In Sect. 4, we present recent work related to our approach and discuss its differences. Finally, we conclude this paper by an outlook on our next steps in Sect. 5.

2 Marrying Model-Driven Engineering and Process Mining

In this section, we briefly describe the main building blocks of both, MDE as well as PM, necessary for the context of this paper, before we present a unifying architecture for their combined but loosely-coupled usage.

2.1 Prerequisites

Model-Driven Engineering (MDE). In each phase of a MDE-based development process “models” (e.g., analysis models, design models) are (semi-) automatically generated by model-to-model transformations (M2M) that take as input models that were obtained in one of the previous phases. In the last
step of this process the final code is generated using model-to-text transformation (M2T) from the initial model [3]. These transformation engineering aspects are based on the metamodels of the used modeling language, which provide the abstract syntax of that language. This syntax guarantees that models follow a clearly defined structure. In addition, it forms the basis for applying operations on models (e.g., storing, querying, transforming, checking, etc.).

As described in [3], the semantics of a modeling language can be formalized by giving (i) denotational semantics by defining a mapping from the modeling language to a formal language, (ii) operational semantics by defining a model simulator (i.e., implementing a model execution engine), or (iii) giving translational semantics by defining, e.g., a code generator for producing executable code. In order to generate a running system from models, they must be executable. This means that a model is executable when its operational semantics is fully specified [3]. However, executability depends more on the used execution engine than on the model itself. The main goal of MDE is to get running systems out of models.

In our approach, we consider executable modeling languages which explicitly state “what” the runtime state of a model is as well as all possible events that can occur during execution [17]. These executable modeling languages not only provide operational semantics for interpreters, but also translational semantics in form of code generators to produce code for a concrete platform to realize the system.

**Process Mining (PM).** PM combines techniques from data mining and model-driven Business Process Management (BPM) [23]. In PM, business processes are analyzed on the basis of event logs. Events are defined as process steps and event logs as sequential ordered events recorded by an information system [8]. This means that PM works on the basis of event data instead of prescriptive models. The main challenge of PM is to capture behavioral aspects. Thereby, specialized algorithms (e.g., the \( \alpha \)-algorithm) produce a Petri net which can be easily converted into a descriptive model in form of a process model. To put it in a nutshell, there is a concrete, running system which is producing logs and there are algorithms used to compute derived information from these logs. Generally in PM, event logs are analyzed from a process-oriented perspective using general modeling languages (e.g., UML, Petri nets) [24].

There are three main techniques in PM: (i) the discovery technique by which a process model can be automatically extracted from log data [23], (ii) the conformance checking technique, which is used to connect an existing process model with an event log containing data related to activities (e.g., business activities) of this process [18], and (iii) the enhancement technique which is used to change or extend a process model by modifying it, or by adding a new perspective to this model [23].

Orthogonal to the dimension of these techniques, there exists a dimension of different perspectives [23]: (i) the control-flow perspective reflects the ordering of activities, (ii) the organizational perspective focuses on resources, organisational
units and their interrelations, (iii) the case perspective deals with properties of individual cases, or process instances, and (iv) the time perspective focuses on execution time analysis and the frequency of events. These perspectives give a complete picture of the aspects that process mining intends to analyze. In [19], van der Aalst suggests to combine perspectives in order to create simulation models of business processes based on runtime information.

In recent work, van der Aalst already brings together PM with the domain of software engineering. For instance in [25], the authors present a novel reverse engineering technique to obtain real-life event logs from distributed software systems. Thereby, PM techniques are applied to obtain precise and formal models, as well as to monitor and improve processes by performance analysis and conformance checking. In the context of this paper we focus on the control-flow and time perspectives of PM.

2.2 Unifying Conceptual Architecture

In this section, we combine MDE with PM by presenting a unifying conceptual architecture. The alignment of these two different research fields may help us, e.g., to verify if the mapping feature of design models is really fulfilled, or if important information generated at runtime is actually missing in the design (i.e., prescriptive) model.

Figure 1 presents an overview of this architecture. On the left-hand side there is the prescriptive perspective, where we use models for creating a system, whereas on the right-hand side there is the descriptive perspective, where models are extracted from running systems (i.e., executed models). In the following, we describe Fig. 1 from left to right.

![Fig. 1. Unifying conceptual architecture for MDE and PM.](image-url)
The starting point is the design language specification at the metamodeling level which defines the syntax as well as semantics of a language like UML, SysML, or a certain domain specific language (DSML). The design model at the modeling level describes a certain system for a specific purpose and has to conform to the chosen design language (see Fig. 1, «conformsTo»). In our approach, such a model describes two different aspects of the system: (i) the static aspect which describes the main ingredients of the domain to be modeled, i.e., its entities and their relationships, and (ii) the dynamic aspect which describes the behavior of these ingredients in terms of events and interactions that may occur among them. For the vertical transition from the modeling level to the realization level (i.e., the process of transforming models into source code), we use code generation at the automation level as introduced in [3]. Finally, at the realization level, the running software relies on a specific platform for its execution (e.g., a Raspberry Pi as presented in our case study in Sect. 3).

At the right-hand side of Fig. 1 (at the top right), we present a logging metamodel—the so-called observation language. This metamodel defines the syntax and semantics of the logs we want to observe from the running system. In particular, we derive this metamodel from the operational semantics of the design language. This means that the observation metamodel can be derived from any modeling language that can be equipped with operational semantics. Figure 1 indicates this dependency at the metamodel level by the dashed arrow and the keyword «refersTo». The observation language has an influence on the code generator, which produces not only the code for the system to run, but also logging information (see Fig. 1, arrow from the observation language (input) to the code generator (output)). This means that the observation language determines which runtime changes should be logged and the code generator provides the appropriate logging code after every change (e.g., state change, attribute value change). Finally, these execution logs are stored as so-called observation models (see Fig. 1, arrow from the execution platform to the observation models). These observation models, which conform to the observation language, thumb the logs at runtime and provide these logs as input for any kind of tools used for checking purposes, e.g., for checking non-functional properties like performance, correctness, appropriateness. For instance, we transform the design language-specific observation model to a workflow representation which can be read by PM analysis tool as presented in our case study.

3 Case Study: Execution-Based Model Profiling

In this section, we perform an exploratory case study based on the guidelines introduced in [20]. The main goal is to evaluate if current approaches for MDE and PM may be combined in a loosely-coupled way, i.e., both can stay as they are initially developed, but provide interfaces to each other to exchange the necessary information to perform automated tasks. In particular, we report on our results concerning a fully model-driven engineered traffic light system which
is enhanced with execution-based model profiling capabilities. All artifacts of the case study can be found on our project website\(^1\).

### 3.1 Research Questions

As mentioned above, we performed this study to evaluate the feasibility and benefits of combining MDE and PM approaches. More specifically, we aimed to answer the following explanatory research questions (RQ) composed of two requirement satisfaction questions (Transformability, Interoperability), an effect question (Usefulness), and a trade-off question (Timeliness):

1. **RQ1—Transformability**: Is the operational semantics of the modeling language rich enough to automatically derive observation metamodels?
2. **RQ2—Interoperability**: Do observation metamodels satisfy interoperability by fulfilling the requirements of existing process mining formats?
3. **RQ3—Verifiability**: Are the generated model profiles resulting from the observation model sufficient for runtime verification?
4. **RQ4—Timeliness**: Are there significant differences between timing of transitions on the specification level and the implementation level?

### 3.2 Case Study Design

**Requirements.** As an appropriate input to this case study, we require a system which is generated by a MDE approach and equipped with an executable modeling language. This means that its syntax and operational semantics are clearly defined and accessible. Furthermore, the approach has to provide translational semantics based on a code generator which may be extended by additional concerns such as logging. Finally, the execution platform hosting the generated code must provide some means to deal with execution logs.

**Setup.** To fulfill these case study requirements, we selected an existing MDE project concerning the automation controller of a traffic light system. We modeled this example by using a small sub-set of UML which we named *Class/State Charts (CSC)* language. CSC stands for UML class diagram and UML state machine diagram, both shown in Fig. 2. The class diagram represents the static aspect of the system, whereas the state machine diagram describes the dynamic one. Generally, UML class diagrams consist of *classes* with *attributes*, and state charts containing *state machines* with *states* and *transitions* between them [21]. In a state chart diagram, *transitions* can be triggered by different types of *events* like signal event, time event, call event, or change event [21]. Both, states and transitions can call *actions*.

Figure 2 presents the class diagram and state machine diagram of the traffic light system modeled in CSC. This system consists of several components such as lights (green, yellow, red) for cars and pedestrians, a controller as well as

\(^1\) [http://www.sysml4industry.org/?page_id=722.](http://www.sysml4industry.org/?page_id=722.)
Fig. 2. CSC class diagram and state machine diagram of the traffic light system.

A blink counter for the pedestrian light. While the CSC state machine diagram (see Fig. 2, on the right-hand side) shows all possible and valid transitions/states within this example, the CSC class TrafficLightController (see Fig. 2, on the left-hand side) specifies the blink counter bc:int=0 and the different lights which can be on or off.

We employed the Enterprise Architect\(^2\) (EA) tool to model the CSC class and state machine diagram. Additionally, we used and extended the Vanilla Source plug-in of EA to generate Python code from the executed CSC (design) models. The code can be executed on a single-board computer. For this purpose we used Raspberry Pi (see Fig. 3, at the bottom left) as specific execution platform. It has to be noted that we aimed for full code generation by exploiting a model library which allows to directly delegate to the GPIO module (i.e., input/output module) of the Raspberry Pi.

3.3 Results

In this subsection, we present the results of applying the approach presented in Sect. 2.2 for the given case study setup. Firstly, we describe the technical realization of the example. Subsequently, we present the appropriate observation metamodel referring to the CSC design language and its conforming observation model. Finally, we generate different model profiles on the basis of PM techniques for checking purposes.

Technical Realization at a Glance. The execution logs of the running code on the Raspberry Pi form the basis for the experimental frame of our approach. Figure 3 gives an overview of its implementation. We extend the code generator to produce Python code (CSC2Python) which enables us to report logs to a log recording service implemented as MicroService, provided by an observation model repository. For data exchange between the running system and the log recording service we used JSON. This means that the JSON data transferred to the MicroService is parsed into log entry elements in the repository. We used the NoSQL database Neo4EMF\(^3\) to store the execution logs for further

\(^3\) http://www.neoemf.com.
analysis. To be able to use established PM tools, we generated XML files from the recorded execution logs (i.e., the observation models).

For the case study of our approach we used ProM Lite 1.1\textsuperscript{4} which is an open source PM tool. Files that this tool takes as input have to correspond to the XSD-schema of the workflow log language MXML\textsuperscript{5}. To accomplish this we used the ATLAS transformation language (ATL)\textsuperscript{[12]} for transforming the observation models to MXML-conform XML files (Observation2WF). In particular, we reverse-engineered the XML Schema of the MXML language into a metamodel. This step enabled us to translate the language-specific observation model into workflow instances (WF Instances) to directly import these instances in ProM Lite. For our case study example the used MXML format was sufficient. Nevertheless XES is the current standard, therefore, we will build on the XES format in future work.

The CSC Observation Metamodel. According to PM techniques, we consider an observation model as an event log with a start and end time registered as a sequences of transactions that having already taken place. However, we do not receive event logs from an executed process model (i.e., the activities of a business process in an ordered manner), rather we receive the traces from transformed log messages of an embedded system. Figure 4 shows the observation metamodel derived from the operational semantics of the CSC design language used in the context of this case study. The figure illustrates that changes at runtime are basically value updates for attributes of the CSC class diagram as well as updates concerning the current active state and current fired transition of the CSC state machine diagram.

\textsuperscript{4} http://www.promtools.org/doku.php?id=promlite.
\textsuperscript{5} http://www.processmining.org/WorkflowLog.xsd.
As shown in the upper section of Fig. 4, these elements are marked with the «observe» stereotype. The CSC dependent observation metamodel is shown in the lower section of Fig. 4. The class Log represents a logging session of a certain running software system with a registered observationStart and an observationEnd. The class Log consists of process instances related to the CSC StateMachine. Every ProcessInstance has a unique id, startTime, and endTime attributes and consists of log entries with the attributes id and timeStamp for ordering purpose (i.e., indicating when the entry was recorded).

Additionally, we defined a subset of a state machine by indicating the stereotypes «case_start» and «case_end». These stereotypes have to be annotated in the design model whenever objects may execute more than one case. The reason for such a stereotype annotation is that, in contrast to business processes, state machines do not necessarily have a clearly defined start- and end point, like in the case of our traffic light system example. This is due to the fact that state machines are often defined for long-life (persistent) objects. This means that only values of objects change over time, but not the objects themselves. Therefore, we defined these stereotypes in our metamodel which enables us to capture single cycles (like cases in PM) of the state machine to be profiled. In our case study example, the start point and end point coincide. When the example starts, there is a safety state only entered once. Each further cycle starts and ends with the state Car→green (see Fig. 2).

The LogEntry either registers an AttributeValueChange, a CurrentStateChange, or a TransitionFiring. CurrentStateChange and TransitionFiring are associated with the state and the transition of the CSC design language. AttributeValueChange has an association with the changing attribute of a class and includes its currentValue.
**Generated Model Profiles.** We used ProM Lite for generating different model profiles from the observation model of the running code. For this purpose we employed ATL model transformations to import the CSC language-specific observation model as input into ProM Lite. By doing so, we focused on two PM-perspectives, (i) the control-flow perspective and (ii) the time perspective (cf. Sect. 2), as well as a (iii) data manipulation one. In the control-flow perspective, we employed the \( \alpha^{++} \)-algorithm of ProM Lite to generate Petri nets for reflecting all attribute value changes as well as state changes and their structure. For profiling the time perspective, we mined the sequence of fired transitions among all states with the inductive miner of ProM Lite and replayed the logs on the discovered Petri net by using a special performance plug-in of this tool.

In a first step of our case study, we implemented a model transformation in ATL which considered the state occurrences (CurrentStateChange) of the running system. By this, we checked on the one hand if the CSC state machine diagram is realized by the code generator as intended (see Fig. 5), and on the other hand, if the state machine executes the specified control-flow on the realization level. This enables, both, a semantically as well as syntactically “equivalence” checking of the prescriptive (design) model and the descriptive (operational) model. In particular, for semantically checking we compared the state space of the state machine with the state space of the profiled Petri net. As shown in Fig. 5 (see the dashed arrows) places with the same targets were merged. The dashed arrow at the bottom right symbolizes a manually interruption of a case. The figure shows that the places and transitions of the Petri net are equivalent to the states and transitions of the CSC state machine diagram presented in Fig. 2. For syntactically checking purpose we may define bi-directional transformation rules to check the consistency [5].

In a second step, we implemented a Python component in order to simulate random system failures which were not reflected in the initial design model presented in Fig. 2. We observed the control-flow perspective of this extended system and found out that the randomly simulated failure states were correctly detected by ProM Lite (compare the Petri net shown in Fig. 6 with that one.

![Fig. 5. Model profile of state changes.](image-url)
Fig. 6. Model profile of state changes including a failure state.

Fig. 7. Model profile of the attribute value changes for the blink counter (bc).

of Fig. 5). Thereby, we proof the usefulness of the approach for runtime verification. It shows that failures which may happened in the implementation phase would be correctly detected and visualized. For instance, this provides useful insights in the running system for validating the code generator and manual code changes.

In a next step, we developed another ATL transformation to extract for each attribute a workflow instance that contains the sequence of AttributeValueChanges. By this, we extracted the shape of the values stored in the attribute to enrich the model with this kind of information and to check if certain value constraints were fulfilled during execution. For instance for the blink counter attribute, we derived a profile which explicitly shows a loop counting from zero to six as depicted in Fig. 7. These logged value changes conform to the attribute (bc) of the class TrafficLightController as shown at the left hand sight of Fig. 2.

In the CSC state machine diagram the timing component is explicitly assigned to transitions (see Fig. 2, «case_start» and «case_end»). In a last step of our case study, we observe the time perspective. Therefore, we needed an additional ATL transformation for filtering the sequence of TransitionFirings (see Fig. 4 from the upper section to the lower section). This sequence includes several iterations of the traffic light system and is used as an input for the performance plug-in of ProM Lite. Our simulation covered 78 cycles, which took 22.26 min, and computed descriptive statistical values for performance evaluation like minimum, maximum and average transition time and sojourn time (i.e., waiting time), as well as the throughput which is the maximum rate at which a system can be processed. Table 1 presents the outcome of this descriptive analysis. To count several cycles (i.e., cases), we annotated the state Car→green
Table 1. Outcome of the performance evaluation based on transition firings.

<table>
<thead>
<tr>
<th>Timing property</th>
<th>Min</th>
<th>Max</th>
<th>Avg</th>
<th>Std.Dev</th>
<th>Freq</th>
</tr>
</thead>
<tbody>
<tr>
<td>Throughput_time</td>
<td>0.00 ms</td>
<td>0.00 ms</td>
<td>0.00 ms</td>
<td>0.00 ms</td>
<td>78</td>
</tr>
<tr>
<td>Waiting_time</td>
<td>2.02 s</td>
<td>2.12 s</td>
<td>2.04 s</td>
<td>19.24 ms</td>
<td>78</td>
</tr>
<tr>
<td>Sojourn_time</td>
<td>2.02 s</td>
<td>2.12 s</td>
<td>2.04 s</td>
<td>19.24 ms</td>
<td>78</td>
</tr>
<tr>
<td>Observation_period</td>
<td>22.26 min</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

with the stereotypes «case_start» and «case_end» as introduced in the CSC metamodel. On average the transition from car yellow to car red is 2.04 s, which is very close to the timing of transition (2 s) of the CSC state machine presented in Fig. 2.

3.4 Interpretation of Results

Answering RQ1. The operational semantics could be transferred into an observational viewpoint. By generating a change class for every element in the CSC design metamodel which is annotated with the «observe» stereotype, we are able to provide a language to represent observations of the system execution. This language can be also employed to instrument the code generator in order to produce the necessary logging statements as well as to parse the logs into observation model elements.

Answering RQ2. By developing ATL transformations from the language-specific observation metamodels to the general workflow-oriented formats of existing PM tools, we could reuse existing PM analysis methods for MDE approaches in a flexible manner. Not only the state/transition system resulting from the state machine can be checked between implementation and design, but also other mining tasks may be achieved such as computing value shapes for the given attributes of the CSC class diagram. Thus, we conclude that it is possible to reuse existing formats for translating the observations, however, different transformations may be preferred based on the given scenario.

Answering RQ3. For runtime verification, we took as input transformed event logs (i.e., selected state changes as a workflow file) and employed the $\alpha++$ algorithm of ProM Lite to derive a Petri net. This generated Petri net, as shown in Fig. 5, exactly corresponds to the state machine, as shown in Fig. 2 on the right hand side. We are therefore convinced that the state machine is realized by the code generator as intended. Similarly, we have done this for attribute value changes. As output we extracted a value shape $[0..6]$ stored in the attribute blink counter (see Fig. 7). Thus, we are also able to enrich the initial CSC class diagram presented in Fig. 2 with runtime information in terms of model profiles. Finally, we manually implemented random failure states in the Python code (not in the
design model) in order to show that these system down states are reflected in the generated Petri net. By applying bi-directional transformations, these additional states may be also propagated to the initial CSC state machine diagram (i.e., prescriptive model) for completing the specification for error-handling states that are often neglected in design models [6].

Answering RQ4. For the detection of timing inconsistencies we filtered the sequence of transitions using an ATL transformation and analyzed it with the performance plug-in of ProM Lite. The inconsistencies between the specification and implementation levels are within the range of milliseconds. The average values of the delays can be propagated back to the design model in order to make the timing more precise during the system execution. The information about timing inconsistencies is especially relevant for time critical and safety critical systems, since this information may mitigate potential consequences of delays. However, it is important to observe a system for a sufficiently long period of time to have enough runtime information for reliable statistical values.

3.5 Threats to Validity

To critically reflect our results, we discuss several threats to validity of our study. First, in the current realization of our approach we do not consider the instrumentation overhead which may increase the execution time of the instrumented application. Of course, this may be critical for timed systems and has to be validated further in the future. Second, the current system is running as a single thread which means we are not dealing with concurrency. Extensions for supporting concurrency may result in transforming the strict sequences in partially ordered ones. Third, we assume to have a platform which has network access to send the logs to the micro service. This requirement may be critical in restricted environments and measurements of network traffic have to be done. Finally, concerning the generalizability of the results, we have to emphasize that we currently only investigated a single modeling language and a single execution platform. Therefore, more experiments are needed to verify if the results can be reproduced for a variety of modeling languages and execution platforms.

4 Related Work

We consider model profiling as a very promising field in MDE and as the natural continuation and unification of different already existing or emerging techniques, e.g., data profiling [1], process mining [23], complex event processing [15], specification mining [6], finite state automata learning [2], as well as knowledge discovery and data mining [9]. All these techniques aim at better understanding the concrete data and events used in or by a system and by focusing on particular aspects of it. For instance, data profiling and mining consider the information stored in databases, while process mining, FSA learning and specification mining focus on chronologically ordered events. Not to forget models@run.time,
where runtime information is propagated back to engineering. There are several approaches for runtime monitoring. Blair et al. [4] show the importance of supporting runtime adaptations to extend the use of MDE. The authors propose models that provide abstractions of systems during runtime. Hartmann et al. [10] go one step further. The authors combine the ideas of runtime models with reactive programming and peer-to-peer distribution. They define runtime models as a stream of model chunks, like it is common in reactive programming.

Currently, there is emerging research work focusing on runtime phenomena, runtime monitoring as well as discussing the differences between descriptive and prescriptive models. For instance, Das et al. [7] combine the use of MDE, run-time monitoring, and animation for the development and analysis of components in real-time embedded systems. The authors envision a unified infrastructure to address specific challenges of real-time embedded systems’ design and development. Thereby, they focus on integrated debugging, monitoring, verification, and continuous development activities. Their approach is highly customizable through a context configuration model for supporting these different tasks. Szvetits and Zdun [22] discuss the question if information provided by models can also improve the analysis capabilities of human users. In this context, they conduct a controlled experiment. Van der Aalst et al. [19] show the possibility to use runtime information and automatically construct simulation models based on event logs. These simulation models can be used, e.g., to evaluate performance of different alternative designs prior to roll-out. Heldal et al. [11] report lessons learned from collaborations with three large companies. The authors conclude that it is important to distinguish between descriptive models (used for documentation) and prescriptive models (used for development) to better understand the adoption of modeling in industry. Last but not least, Kühne [13] highlights the differences between explanatory and constructive modeling, which give rise to two almost disjoint modeling universes, each of it based on different, mutually incompatible assumptions, concepts, techniques, and tools.

5 Conclusion and Future Work

In this paper, we pointed to the gap between design time and runtime in current MDE approaches. We stressed that there are already well-established techniques considering runtime aspects in the area of PM and that it is beneficial to combine these approaches. Therefore, we presented a unifying conceptual architecture for execution-based model profiling, where we combined MDE and PM. We built the approach upon traditional activities of MDE such as design modeling, code generation, and code execution. In the conducted case study, we demonstrated and evaluated this approach on the basis of a traffic light system example. While the first results seem promising, there are still several open challenges, which we discussed in the threats to validity in the case study section. As next steps, we will focus on the observation of further PM perspectives (e.g., the organisational perspective) that can be used for software component communication discovery and on the reproduction of our current results by conduction additional case
studies, in this respect, domain-specific modeling languages (DSMLs) would be of special interest.

Acknowledgment. The authors are affiliated with the Christian Doppler Laboratory for Model-Integrated Smart Production (CDL-MINT) at TU Wien, funded by the Austrian Federal Ministry of Science, Research, and Economy (BMWFFW) and the National Foundation of Research, Technology and Development (CDG). Furthermore, the authors would thank LieberLieber Software GmbH for the provisioning of the traffic light example.

References


12 Reverse Engineering of Production Processes based on Markov Chains

A. Mazak, P. Patsuk-Bösch and M. Wimmer;
DOI: 10.1109/COASE.2017.8256182
Reverse Engineering of Production Processes based on Markov Chains

Alexandra Mazak¹, Manuel Wimmer¹ and Polina Patsuk-Boesch¹

Abstract—Understanding and providing knowledge of production processes is crucial for flexible production systems as many decisions are postponed to the operation time. Furthermore, dealing with process improvements requires to have a clear picture about the status of the currently employed process. This becomes even more challenging with the emergence of Cyber-Physical Production Systems (CPPS). However, CPPS also provide the opportunity to observe the running processes by using concepts from IoT to producing logs for reflecting the event happening in the system during its execution. Therefore, we propose in this paper a fully automated approach for representing operational logs as models which additionally allows analytical means. In particular, we provide a transformation chain which allows the reverse engineering of Markov chains from event logs. The reverse engineered Markov chains allow to abstract the complexity of run-time information as well as to enable what-if analysis whenever improvements are needed by employing current model-based approaches and measurement-based technologies. We demonstrate the approach based on a lab-sized transportation line system.

I. INTRODUCTION

Production systems are becoming more and more software-intensive, thus, turning into Cyber-Physical Production Systems (CPPS). In the manufacturing environment, these CPPS comprise smart machines, storage systems and production facilities capable of autonomously exchanging information and triggering actions [1]. As a consequence, the complexity of CPPS is continuously increasing. To deal with this increased complexity, modeling is a promising approach in this context. However, current modeling foundations and practices are still lacking behind the emerging requirements of Industry 4.0. Amongst others, models are considered as static entities, mostly used as blueprints in the early design phase, but they are basically neglected in later lifecycle phases, which drastically limits their value during the production systems’ operation [2]. This gap may result in discrepancies between design time models and its real world correspondent [3]. Therefore, one of the major milestones is the backpropagation of run-time information (i.e., measured data) derived from operations to engineering artifacts by exploiting Internet of Things (IoT) concepts, like smart sensors and actuators [1]. These requirements are emerging in industrial automation systems engineering, and thus, the software engineering community has been confronted with them [4].

In recent years, there emerged modeling technologies (e.g., multi-viewpoint modeling, multi-paradigm modeling) which allow users to apply different techniques such as simulation, model checking, etc., by using very promising modeling formalisms. However, modeling is mostly exploited for code generation approaches [5]. The other benefits of modeling such as dynamically extracting run-time models to better link operations with engineering are mostly overlooked. To counteract, Model-driven Engineering (MDE) has been proposed by which models are developed on a higher level of abstraction [6], [7], [8]. MDE has already found its way into the systems engineering domain, but not necessarily into the domain of industrial automation systems engineering [5].

In MDE, models are mostly used in prescriptive ways for system engineering. Although, models are an important ingredient to realize a system, for later phases in the systems’ life-cycle additional model types are beneficial to use [9]. Therefore, descriptive models may be employed to better understand how a system is actually realized and how it is operating in certain environments [10]. Compared to prescriptive models, these other types of models are only marginal explored in the field of MDE, and if used at all, they are built manually. Thus, we aim for an automated reverse engineering approach that combines model-based downstream information derived from prescriptive models and measurement-based upstream information of a running production for managing the complexity of CPPS.

The remainder of this paper is structured as follows. In Section II, we outline the state of the art and main building blocks in the context of our approach. The case study used for motivating and demonstrating this approach is presented in Section IV. The reverse engineering framework is described in Section III. The application of our approach for the case study and an evaluation with a critical discussion are presented in Section V. Concluding the paper, in Section VI, remaining steps are discussed and an outlook on future research work is given.

II. STATE OF THE ART

In this section, we give a brief overview of related work in the field of industrial engineering, and we briefly describe the theoretic background and main building blocks, necessary for the context of the introduced approach which is influenced by these various different research fields. It is our goal to merge these topics as introduced in Section III.

A. Model-driven Engineering Techniques

There are two major MDE techniques: (i) metamodeling for specifying modeling languages, i.e., the structure and content of valid models, and (ii) model transformations to systematically manipulate models [7]. In our approach,
we use metamodels to specify language concepts and their relationships (i.e., abstract syntax), as well as concrete syntax (i.e., model notation), and semantics [7]. Metamodeling environments allow to generate modeling environments and are providing generic tool support, which can be employed for all the modeling languages defined with a metamodeling environment.

Generally, a model transformation is a program executed by a transformation engine which takes one or more models as input to produce one or more models as output. Model transformations are used to solve different tasks like code generation, model refactoring, or reverse engineering to name just a few examples. An important aspect is that model transformations are developed on the metamodel level, and thus, are reusable for all valid model instances [7]. For instance, we use Model-driven Reverse Engineering (MDRE) to create a set of models that represent a system. These models can then be used for different purposes, e.g., metrics and quality assurance computation, tailored system viewpoints, etc. For more insights in model-driven engineering in practice, we refer the interested reader to the work of Brambilla et al. [7].

B. Process Mining

Process mining (PM) is a process-centric management technique bridging the gap between data mining and traditional model-driven Business Process Management (BPM) [11], [12]. The main objective of PM is to extract valuable, process-related information from event logs for providing detailed information about actual processes, for instance, to identify bottlenecks, to anticipate problems, to record policy violations, to streamline processes, etc. [12]. In PM, events are defined as process steps and event logs as sequential events recorded by an information system [13]. This demonstrates that unlike BPM approaches PM works on the basis of event data instead of design models (i.e., prescriptive models).

In [11], van der Aalst lists three basic PM goals, which are (i) discovery, (ii) conformance, and (iii) enhancement. Discovery means to take an event log as input and to produce a process model as output. When targeting for conformance an existing process model is compared with an event log of the same process. This means an event log and a model are used as input and diagnostic information is produced as output. Thereby, a user can check whether information recorded in the log conforms to the intended model and vice versa. The third type of PM is called enhancement. Its idea is to improve or extend an existing model. It takes an event log and a model as input and produces a new model as output.

Current event processing technologies usually monitor single streams of events at a time. Even if users monitor multiple streams, they often end up with multiple “silos” views. A more unified view is needed that correlates with events from multiple data streams of various sources and in different formats. Thereby, heterogeneity and incompleteness of data are major challenges [14]. Mostly, PM operates on the basis of events that belongs to cases that are already completed [15]. This off-line analysis is not suitable for cases which are still in the pipeline. In [11], the author mixes current data with historic data to support on-line and off-line analysis.

C. Run-time Models

There are several different approaches for run-time modeling. All of them aim on bridging the gap between design time modeling and run-time modeling to enable run-time analysis. Blair et al. [16] show the importance of supporting run-time adaptations to extend the use of MDE. They propose models that provide abstractions of systems during run-time. These operational models are an abstraction of run-time states. Due to this abstraction, different stakeholders can use the models in various ways, like dynamic state monitoring or observing run-time behavior.

Hartmann et al. [17] go one step further. They combine the ideas of run-time models with reactive programming and peer-to-peer distribution. The authors define run-time models as a stream of model chunks, like it is common in reactive programming. The models are continuously updated during run-time, therefore, they grow indefinitely. With their interpretation that every chunk has the data of one model element, they process them piecewise without looking at the total size. In order to prevent the exchange of full run-time models, peer-to-peer distribution is used between nodes to exchange model chunks. In addition, automatic reloading mechanism are used to respond on events for enabling reactive modeling. As the models are distributed, operations like transformations have to be adapted. For this purpose transformations on streams as proposed by Cuadrado et al. can be used [18]. Reactive programming aims on enabling support for interactive applications, which react on events by focusing on streams. For this purpose a typical publish/subscribe pattern, well known as the observer pattern in software engineering [19], is used. Khare et al. show the application of such an approach in the IoT domain in [20].

D. Industrial Engineering

Folmer et al. [21] present in their work a valve diagnosis system (VDS) using data aggregated from multiple sources across company borders. The authors introduce a usual model for valve behavior and an adapted model enriched by process data and detailed design data of a plant’s equipment. In their approach, they combine model-based fault detection and isolation (FDI) for valves with measurement-based techniques based on industry standards. The goal is to identify differences between both models for detecting gradual increasing valves’ faults for plug wear and plug contamination.

Danar et al. [22] present approaches for process analysis and organizational mining in the domain of production automation engineering. The main goal of their mining approach is to align the run-time system process model with the design process model for conformance checking like in PM, as well as, to analyze the structure and interactions of system components during run-time, e.g., for improving maintenance planning. The authors evaluate their approach
by using the SAW simulator [23], based on the Simulator for Assembly Workshops (SAW) project, as a running use case for a production automation system.

Vogel-Heuser et al. [24] focus in their work on software evolution in the domain of automated production systems (aPS). They investigate the evolution and co-evolution of engineering models and code, quality assurance, as well as, variant and version management. In their work, the authors point to the fact that only focusing on challenges regarding the evolution of long-living automated production systems from a software perspective is not sufficient. Therefore, they (i) determine "why" this is not sufficient, (ii) present approaches to address the challenges in the aPS domain, (iii) define research goals, and (iv) identify "how" the presented approaches can lead to synergetic research goals and results focusing on the evolution of long-living aPS.

In [25], Vogel-Heuser et al. provide an open case study for studying the evolution of automation systems by a bench-scale manufacturing system called Pick and Place Unit (PPU). They present various scenarios to study the evolution in industrial plant automation and to document it. In [26], the authors present two different case studies. One from the domain of information systems, and the other one from the domain of automated production systems in order to validate their introduced approaches for analyzing the maintainability of software intensive systems.

E. Summary

One may argue that the before mentioned research fields often treated in isolation. Generally, the focus in MDE is on prescriptive models for code generation, whereas that one in PM is on descriptive models. However, Section II-D illustrates that both model types are required for the backpropagation of run-time information to design models in order to keep them up-to-date over their whole life-cycle. The models@run-time approach goes in a similar direction (cf. Section II-C). However, this research field focuses more on the creation of run-time models and their interaction with the environment as well as their continuously update during run-time based on changes within the environment, and not necessarily to combine design time with run-time for real-time tracking and tracing of models for their improvement, like we present in our stochastic-based reverse engineering approach.

III. REVERSE ENGINEERING OF MARKOV CHAINS FROM EVENT LOGS

A. Overview

In this section, we present a model-driven as well as data-driven reverse engineering approach to compute behavioral models from timely observations of system components. For this purpose we combine the prescriptive perspective of MDE with the descriptive one of PM (cf. Section I). By this, we generate descriptive models from execution logs which reflect the de-facto process characteristics and important performance characteristics of a system during run-time. In particular, we observe and analyze activities happening in a system during run-time for providing reasoning mechanisms for future adaptations. In doing so, we observe (i) resources that offer computing capabilities, (ii) workload that describes how these resources are being used, and (iii) workload intensity in terms of arrival times.

This approach bases on two metamodells, namely CETO and MUPOM. CETO extends existing concepts of PM (cf. Section II-B) like discovery process models from event logs. In MUPOM, we employ Markov processes of probability theory to describe resource-specific behaviors. Both metamodells are needed to combine the model-based perspective taking during systems engineering with the measurement-based one taking during the system’s operation. CETO and MUPOM enable us to observe and to analyze the main characteristics of a system at run-time.

B. CETO Metamodel

The Components Emission and Timely Observations (CETO) model captures measurement data of the running system under observation. These tracked observations (i.e., resource-specific operation calls) base on the emissions generated by resources (e.g., machines, specific components) when being used, e.g., in a production process (cf. Section V). These emissions are observed by operational logs which reflect the activities happening in a system (e.g., IAF plant) during run-time. In this respect, these operational logs are handled very similar to event logs of PM (cf. Section II).

In the CETO model, we consider the duration time of operations applied on items. In a running system, these operations result in computing time on resources which is measureable. The time interval for an operation may vary for one item to another (e.g., based on the size of an item), or if an item processes an operation multiple times. The observation of many operation durations allows to build a probability distribution function over these durations. To put it in a nutshell, the CETO model tracks, measures and stores the emissions of the resources when being used during the system’s operation.

C. MUPOM Metamodel

The Markov Usage Process and Operation Measurements (MUPOM) model describes resource-specific behaviors by using the Markov model formalism. We implemented the MUPOM model with the goal to model stochastic processes in order to extract the (probabilistic) workload of a production system. A workload is the primary stochastic element, since it changes and occurs in a non-deterministic way. It is used in a wide range of performance engineering literature in, both, model-based and measurement-based approaches (e.g., [27], [28], [29]).

The workload is the amount of work that requests some sort of service in a specific time interval. We describe the workload of a system by using simple Markov models like Markov chains. For this purpose we construct a model λ given the observing sequence of emissions in a specific time interval Oτ, where the probability Pr(Oτ | λ) is maximized [30]. To achieve this, we have to assume that
the system under observation is ergodic. This means that the system is irreducible, aperiodic, and positive recurrent. We briefly summarize these preconditions as follows [31]:

A system is irreducible, if it is possible to reach each state from any other state. The probability of being in state \( j \) after \( n \)-steps starting from \( i \) must therefore be greater than zero.

\[
P_r(X_n = j \mid X_0 = i) > 0 \quad (1)
\]

A system is aperiodic, if the system state is not system-atically connected to time. And a system is recurrent if all states of the system are recurrent. A state is recurrent, if the summed probability of returning to that state for an infinite number of steps \( n \) is finite. To be positive recurrent, a system must be irreducible and aperiodic. This means that the system periodically restarts itself in finite time and every state \( i \) is visited infinitely often. This equates a expected return of [31]:

\[
E(\min \{ n \geq 1 : X_n = i \}) < \infty \quad (2)
\]

There are multiple ways to use a system. Thus, the workload may be substituted by so-called operational profiles. A single operational profile is expressed by sequences of component activations within the system. In our approach, the time between two component activations is described as “think times”, which we express by probability distributions. Workload intensities are on top of operational profiles and describe the amount of items that are processed by the system in a certain time interval. Similar to the Probabilistic Grammar model (HPG) [32], we take all components of a system (e.g., turntables, conveyors, machines) as states and the relations between them as transitions.

The MUPOM metamodel defines only open workloads with external arrivals and departures. These arrival rates can be modeled by using three possible distributions: Poisson, Exponential, and Normal distribution. Therefore, duration and think times follow one of these distributions.

**D. Transformation Chain**

On the basis of the previously discussed MDE techniques (cf. Section II), we present a modular transformation chain from observed logs to Markov chains to abstract complexity of run-time information and for analytical purposes.

Markov chains have a discrete time space that can be finite or infinite. They can occur at any point in time. If the state transition probabilities do not change over time, a Markov chain is stationary. In the case of a discrete state space, the transition probabilities between states can be simply encoded in a transition matrix, whereas in continuous time a transition rate matrix is used [33]. In order to get a \( n \)-steps transition probability of moving from a state \( i \) to an other state \( j \), the transition matrix \( P_s \) for \( n \to \infty \) the resulting matrix may converge to a certain distribution which is called limiting probability, formally defined in [33] as:

\[
\pi_j = \lim_{n \to \infty} P^n_{ij} \quad j \in S 
\]

where \( \pi_j = \min \{ n \geq 1 : X_n = j \} \) and \( \pi_i = E [\pi_j] \).

In order to combine CETO and MUPOM models and to transfer MUPOM to a Markov chain, we apply model-to-model transformations [35]. We apply these transformations to transform (i) an operation topology to CETO, (ii) CETO to MUPOM, and (iii) MUPOM to a Markov chain. This process, which we call transformation chain, enables the reverse engineering of Markov chains from event logs.

In a first step, we implement the CETO model by the OperationAndTraceMonitor tool. This instrumentation tool enables (i) to track an item’s navigation during run-time, (ii) to measure the duration of defined operations, and (iii) to write the observations to a CSV file for analysis purposes. The OperationAndTraceMonitor produces two files: (i) operationDuration.csv and (ii) itemTrace.csv.

In the next step, the MUPOM model approach is implement-ed by the UserTrace2Markov tool. We use the output files (e.g., item traces) of the OperationAndTraceMonitor tool as input of the UserTrace2Markov tool for computing Markov chains. In its first version, this tool constructs a Markov chain by calculating a transition matrix. Therefore, we calculate the average processing time for items based on the transition probability between two states for a concrete example (cf. Section V, Table I). Furthermore, think times of every state are calculated and described by an expected mean value and a standard deviation. Finally, the total aggregated time spent in each state by every item is summed up.

**IV. CASE STUDY**

In this section, we provide the description of a reference case study, a lab-sized production system hosted at IAF of the Otto-v.-Guericke University Magdeburg [36] which is subsequently used as a demonstration example to exemplify our process reverse engineering approach. The IAF production system (cf. Figure 1) consists of a transportation line made up of sets of turntables, conveyors, and multi-purpose machines.

Each turntable is equipped with an inductive proximity sensor for material detection and a motor for table rotation. The transportation line is wired to a modular fieldbus I/O system, which, in turn, communicates with Raspberry Pi based controllers by Ethernet. The Raspberry Pi based controller is running a Programmable Logic Controller (PLC) program governing the transportation line. Such programs logically divide the transportation line in three different areas as depicted in Figure 1. The production plant is supposed to continuously processes items by its multi-purpose machines located in one of the three areas. Turntables and conveyors are in charge of moving such items to these machines. In the given IAF production system different events are observable such as

---

3Models realized for this case study can be downloaded from our CDL-MINT web site at the following address: https://cdl-mint.big.tuwien.ac.at/case-study-artefacts-for-case-2017/
starting and ending different kinds of activities, for instance, turning an item by a turntable, transferring an item by a conveyor, and processing an item by a machine. As we have several turntables, conveyors, and machines, we assume that each of these components has a unique identifier (ID). The presentation of the plant topology is achieved by Figure 2 which uses a SysML block diagram as notation. In the upper part of this figure, the different component types used for the production system are described by the block definition diagram including the operations which are provided by these components. We assume that the start and end events of calling these operations are logged by the controller. In the bottom part of Figure 2, we show the instances of the component types and how they are connected to represent the topology of the system by illustrating these aspects as an internal block diagram. In particular, we present Area 2 as white-box to exemplify the structure of the production system. Even though, some aspects of the system are already explicitly modeled by SysML, a more operational-oriented view is needed to understand the process quality as well as the impact of changes on the behavioral aspects of the system. Of course, one could now start with modeling the intended process which should be supported by the production system with the risk of misinterpreting the system execution. A promising alternative is to observe the events of the running system and to derive process models automatically by applying performance metrics.

V. Evaluation Study

A. Setting and Outcome

For the purpose of validating our approach, we developed a prototypical implementation of the introduced modular transformation chain in the context of the IAF plant case study (cf. Section IV). All artifacts of this evaluation can be found on our CDL-MINT project website.

In particular, we created a SysML-based simulator for the IAF plant case study which is able to produce log files using the OperationsAndTraceMonitor (cf. Section III-D). An excerpt of the output is shown in Listing 1. The log entries are structured as follows: for each component or resource (e.g., turntable, machine, conveyors), identified by path expressions (e.g., systemID/areaID/componentID), we note the time stamp when the resource is requested by a certain item. Thus, this time stamp is considered as the start of a particular resources-specific operation, e.g., Conveyor.transfer(). The end of an operation is not explicitly represented in the log file since it is derived by searching for the next log entry (having the time stamp as close to the given time stamp for the start operation) for the given item. The textual log file format is inspired by existing log formats as used, e.g., for Web servers. However, please note that this format can be automatically parsed in a model structure to allow compatibility, for instance, with different process mining tools, like ProMLite.

Listing 1. Exemplary itemTrace log file excerpt for the IAF plant case study

```
#Fields : component,timestamp,item
entered : /IAF/a2/t1,2017-02-08-23-28-51,923b4f191d48
entered : /IAF/a2/c1,2017-02-08-23-28-54,923b4f191d48
entered : /IAF/a2/t2,2017-02-08-23-28-57,85e51557c3e
entered : /IAF/a1/t2,2017-02-08-23-28-61,923b4f191d48
entered : /IAF/a2/c1,2017-02-08-23-28-63,85e51557c3e
entered : /IAF/a2/ml,2017-02-08-23-28-69,923b4f191d48
entered : /IAF/a2/t1,2017-02-08-23-28-74,5b73e47866d4
```

The produced log files are the input for the UserTrace2Markov tool (cf. Section III-D), which produces a transition probability matrix of the underlying Markov chain. We illustrate this step for Area 2 of the IAF plant example in Table I. The transition probability matrix has been generated based on the observed item traces from one state to another as described in Section III-D.
In order to visualize the results of this matrix, as well as the statistical performance information in a graphical modeling language, we additionally developed a specific modeling editor for MUPOM. This editor supports a domain-specific modeling language (DSML) for representing the resulting information to engineers. We present the resulting model for the given case study in Figure 3. This figures shows that there is some backwards routing which seems unnecessary. Since C1 is no longer backwards routing, and consequently, is routing again the items forward. The reason for this is that the items are moved in one iteration from the entry point of the transportation line onwards which may cause backwards routing as the successor components are still occupied. In contrast, if a different routing strategy is used, such as moving items first from the components close to the exit point of the transportation line to make space for new items, the workload of the system can be positively improved.

By applying the proposed approach and its prototypical implementation, production systems provide a means for “self-modeling”. Since structural models are mostly build in the engineering phases to guide the actual construction, performance models may be also employed already in these early phases. Often the experience is missing to approximate concrete numbers about workload characteristics. By using the presented measurement-based part of our approach, we are able to provide concrete information about the production system which may be used for checking workload assumptions made in the engineering phases, as well as, to find improvement possibilities after launching the first version of a system.

### VI. CONCLUSIONS AND OUTLOOK

In this paper, we presented an approach to automatically combine model-based downstream information derived from design (prescriptive) models with measurement-based upstream information derived from a running system. This reverse engineering approach computes behavioral models from timely observations based on the system components emission in form of operational logs (cf. Section III). These logs reflect the activities happening in a system during runtime. In the measuring part of our approach, we consider the duration time of operations applied on items (e.g., resource-specific operation calls). By tracing these logs, we provide the basis to abstract Markov chains by means of the presented modular transformation chain (cf. Section III-D). By this, we are able to deal with the complexity of run-time information and to provide reasoning mechanisms about future adaptations as presented in our evaluation (cf. Section V).

For future lines of research, we consider the following three topics: (i) big data analytics such as clustering of execution logs, e.g., to separate log entries into groups representing the behavior of different processes for testing purposes, (ii) computing complex events of higher abstraction based on the observed traces.

#### TABLE I

Transition Probability Matrix for the Components of Area 2 of the IAF Plant

<table>
<thead>
<tr>
<th>pij</th>
<th>t1</th>
<th>c1</th>
<th>t2</th>
<th>m1</th>
<th>c2</th>
<th>t4</th>
<th>c3</th>
<th>in</th>
<th>out</th>
</tr>
</thead>
<tbody>
<tr>
<td>t1</td>
<td>0</td>
<td>0.8</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.2</td>
</tr>
<tr>
<td>c1</td>
<td>0.1</td>
<td>0</td>
<td>0.9</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.1</td>
</tr>
<tr>
<td>t2</td>
<td>0</td>
<td>0.1</td>
<td>0</td>
<td>0.9</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.1</td>
</tr>
<tr>
<td>m1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.05</td>
<td>0</td>
<td>0.95</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>c2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>t4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.2</td>
<td>0.8</td>
</tr>
<tr>
<td>c3</td>
<td>0.98</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.02</td>
<td>0</td>
</tr>
<tr>
<td>in</td>
<td>0.95</td>
<td>0</td>
<td>0.05</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>out</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Fig. 3. MUPOM model of Area 2 of the IAF production system based on the observed traces.

**B. Critical Discussion**

We only present a particular formalism which may be exploited for providing reasoning of the extracted observations. By using the modular transformation chain, we may also employ other formalisms such as, Petri nets which are automatically producible from the observed logs in the ProMLite tool. By this, interactive visualization may provide further insights on the currently employed production processes. We consider this line as a promising future research line which may complement the presented reverse engineering approach. Furthermore, other types of analysis can be performed. For instance, by utilizing the JMT tool, we are able to provide estimations for the given systems such as the required buffer length. By estimating the arrival rates of items, we are able to compute continuous time Markov chains which describe the probability of a certain buffer usage. For instance, the Markov chain shown in Figure 4 is computed for our given example. Based on this model, we can assume the maximum requirement for the buffer length as 6 given a arrival rate of 0.1 items per minute and a service time of 4 minutes per item.

**VI. CONCLUSIONS AND OUTLOOK**

In this paper, we presented an approach to automatically combine model-based downstream information derived from design (prescriptive) models with measurement-based upstream information derived from a running system. This reverse engineering approach computes behavioral models from timely observations based on the system components emission in form of operational logs (cf. Section III). These logs reflect the activities happening in a system during runtime. In the measuring part of our approach, we consider the duration time of operations applied on items (e.g., resource-specific operation calls). By tracing these logs, we provide the basis to abstract Markov chains by means of the presented modular transformation chain (cf. Section III-D). By this, we are able to deal with the complexity of run-time information and to provide reasoning mechanisms about future adaptations as presented in our evaluation (cf. Section V).

For future lines of research, we consider the following three topics: (i) big data analytics such as clustering of execution logs, e.g., to separate log entries into groups representing the behavior of different processes for testing purposes, (ii) computing complex events of higher abstraction based on the observed traces.

on filtered execution logs, e.g., to analyze the reliability and availability of a system, (iii) the back-annotation of results in existing behavioral (prescriptive) models by means of a language specific model profiling approach.

VII. ACKNOWLEDGMENTS

This work has been funded by the Austrian Federal Ministry of Science, Research and Economy (BMWFW) and the National Foundation for Research, Technology and Development (Fonds der Chemischen Industrie) (FDI). We are grateful for the help and support of Johannes Arntz. Many thanks also to Prof. Dr.-Ing. Arndt Lüder of the Faculty for Mechanical Engineering at Otto-v.-Guericke University Magdeburg, Germany for providing information on the demonstrator case.

REFERENCES


13 Automatic Reverse Engineering of Interaction Models from System Logs

A. Mazak, S. Wolny and M. Wimmer;
DOI: 10.1109/ETFA.2019.8869502
Automatic Reverse Engineering of Interaction Models from System Logs

Sabine Wolny∗, Alexandra Mazak ∗, Manuel Wimmer∗

∗Christian Doppler Laboratory for Model-Integrated Smart Production (CDL-MINT)

Institute for Business Informatics - Software Engineering
Johannes Kepler University Linz, Science Park 3, 4020 Linz, Austria
{firstname}.{lastname}@jku.at

Abstract—Nowadays, software as well as hardware systems produce log files that enable a continuous monitoring of the system during its execution. Unfortunately, such text-based log traces are very long and difficult to read, and therefore, reasoning and analyzing runtime behavior is not straightforward. However, dealing with log traces is especially needed in cases, where (i) the execution of the system did not perform as intended, (ii) the process flow is unknown because there are no records, and/or (iii) the design models do not correspond to its real-world counterpart. These facts cause that log data has to be prepared in a more user-friendly way (e.g., in form of graphical representations) and algorithms are needed for automatically monitoring the system’s operation, and for tracking the system components interaction patterns. For this purpose we present an approach for transforming raw sensor data logs to a UML or SysML sequence diagram in order to provide a graphical representation for tracking log traces in a time-ordered manner. Based on this sequence diagram, we automatically identify interaction models in order to analyze the runtime behavior of system components. We implement this approach as prototypical plug-in in the modeling tool Enterprise Architect and evaluate it by an example of a self-driving car.

Index Terms—Log traces, model transformation, sequence diagram, interaction model

I. INTRODUCTION

Nowadays, in Model-driven Engineering (MDE) the use of object-oriented modeling languages and code generators for generating code is an already established approach for developing complex systems [1]. However, even if systems are described by means of such modeling languages and code generators are used to transform model elements to corresponding code statements, the execution of these statements is typically not represented in the same structure as the languages’ metamodel. Based on this fact it is difficult to prove whether the design model corresponds to its runtime counterpart, meaning that the discovery of discrepancies is not straightforward. Therefore, engineers would benefit if they could treat runtime data in the same way as design models, i.e. operate with them like standard UML or SysML models. This would help to discover discrepancies more easily and, if appropriate, propagate this information back to the initial model. Thus, an automatic control and improvement of design decisions is enabled by establishing the well-known PDCA (plan-do-check-act) management method in the field of MDE.

During the execution of software or the operation of systems, the behavior, communication, as well as executed operations can only be traced based on sensor value streams or logging code. Typically such log traces have the form of huge text-based files, which are difficult to read and process. Therefore, it is not straightforward to fully understand and track the interaction and communication between system components. A scalable reverse engineering approach is needed that automatically transforms log traces to an appropriate graphical representation allowing an object-oriented view on executed operations as well as the back propagation of runtime information to design models.

In this paper we overcome these obstacles (i) by providing an automatically performed text-to-model transformation from text-based log traces to a graphical representation in form of UML sequence diagrams as object-oriented interaction models of executed operations, (ii) by aligning these models with their initial counterpart (i.e., design models) for creating so-called trace links, and (iii) by automatically creating runtime profiles and displaying these profiles in the design model.

The remainder of this paper is structured as follows. Section II shows the background as well as a motivating example to underline the challenges. In Section III, we present our automatically executable reverse approach by presenting a metamodel for object-oriented logs and by describing the architecture for the approach, and its prototypical implementation. Section IV demonstrates the evaluation of the introduced approach based on an example of log traces generated by a self-driving car. Section V discusses related work. In Section VI, we conclude this paper by an outlook on our next steps.

II. BACKGROUND AND MOTIVATING EXAMPLE

After briefly discussing the main background which forms the basis of our approach, we present a motivating example and discuss open issues to address.

In MDE, the abstraction power of models is used to tackle the complexity when representing systems [1]. From an abstracted point of view, MDE follows the principle of “everything is a model” [2]. This means, MDE supports system as well as software engineers by providing formal models like a tool box to achieve simplicity, generality, and integration.
in the design of systems. In this early phase of development such models are used to create and describe the scope and detail of interest. These so-called design models are then used to realize a system by automatically transforming model elements to code statements which can be then executed on a platform, as already mentioned in Section I. For this purpose MDE provides model transformations such as Model-to-Model (M2M) or Model-to-Text (M2T) transformations [2]. Based on this method, we implement a Text-to-Model (T2M) transformation to transform text-based log trace files to a user-friendly representation for analyzing the execution process of a system (cf. Section III). This means that we use established methods and techniques from MDE as background to establish an end-to-end traceability from design to runtime and back.

A. Motivating Example

Consider a simple autonomous car consisting of sensors, motor and servo controls for driving itself (i.e., without human control) in different environmental settings. This car should be able to (i) recognize barriers, (ii) change direction if necessary, (iii) drive forward and backward or stop. In order to fulfill these requirements the car needs an autonomous acting controller that controls the motor as well as the servo control based on continuously gathered sensor value streams. At design time, the structural and behavioral aspects of such a car are modeled by using a modeling language like UML. The components with their properties and operations are modeled in a class diagram (CD) whereas the intended behavior of the system is modeled by a state machine (SM).

As usual in MDE, the formal model of a SM is used to automatically generate code statements for the car controller. This means that the modeled behavior at design time is used for running the controller. However, during operation the car is driving in different environmental settings. In this execution phase, it is important for the engineers to know whether the system behaves as intended or if there are occurring any unexpected transitions or error states.

In addition, it would be of interest if every operation is executed and, if so, how often, and if there are any specific interaction patterns between controller, motor and servo. To obtain such information based on the runtime behavior of the car, message flows between these components has to be logged and analyzed. However, such textual-based log files are huge, and therefore, difficult to interpret for system engineers. The challenges are: (i) providing a method to visualize those logs in a readable format, (ii) querying communication messages between the controller and the actuators (i.e., servo and motor), and (iii) extracting information of the observed system for improving the design after each execution, so to say, for holding the “model-in-the-loop” according to the PDCA cycle (cf. Section I).

III. APPROACH

In this section, we present our model-driven approach for automatically reverse engineering interaction models in terms of sequence diagrams from system logs for enhancing design models with runtime views. We start this section by presenting the requirements for our architecture and the required structure of system logs in terms of a metamodel. Subsequently, we present the conceptual architecture, and finally, the prototypical implementation of our approach.

A. System Requirements

First of all, we have a number of prerequisites to be met: (i) the structure of the system and its behavior has to be expressible by means of a family of object-oriented modeling languages such as UML class diagrams and UML state machines, (ii) the executed operations have to be logged and should represent inter-object communication, and (iii) the different executed operations have to be uniquely identifiable to ensure that the system logs can be clearly mapped to classes and their assigned operations.

B. System Logs - The Object-Oriented View

<table>
<thead>
<tr>
<th>Case</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>String</td>
</tr>
<tr>
<td>kind</td>
<td>{REQ, RES, ASY}</td>
</tr>
<tr>
<td>message</td>
<td>String</td>
</tr>
</tbody>
</table>

Fig. 1. Metamodel for an object-oriented system log representation

On the basis of the observed and recorded system logs, we automatically reverse engineer an object-oriented interaction model making the communication among system components read- and traceable, as well as enabling the back propagation of profiled information to the design models. With the term object-oriented interaction model we refer to UML interaction models. Such models follow the object-oriented paradigm: objects communicate with other objects via messages. A sequence of different messages results in an interaction between a set of objects. One kind of displaying such interaction models are UML sequence diagrams[2]. Amongst others, sequence diagrams focus on the representation of different interaction participants with their lifelines and messages between them, which can be either synchronous or asynchronous, based on their temporal order. For reconstructing such interaction models we develop a metamodel for representing system logs in an object-oriented approach.

1http://www.uml.org/

2https://www.uml-diagrams.org/sequence-diagrams.html
manner. This enables us to analyze logs from an object-oriented viewpoint as an explicit model. System logs are explicitly expressed as models and thus the unification power of models [2] can be fully applied. For example, to compare two system logs, model-oriented approaches such as model comparisons and model differencing processes can be used without taking further action.

Fig. 1 shows the metamodel we employ for capturing object-oriented system logs. A Log consists of any number of Cases which in turn consist of any number of interactions. Such an Interaction has to be composed of a Sender, a Receiver, and a Message. Sender and Receiver refer to an Object and can have additional optional Feature Values. A Message can comprise of any number of Parameter Values. All elements of this object-oriented log (except values) are indirect instances of IdentifiableElement, and therefore, have an id assigned. In addition, this object-oriented view can be mapped to a class view where Objects are referencing Classes, Feature Values are referencing Features, Messages are referencing Operations, and finally, Parameter Values are referencing Parameters. The presented metamodel enables the construction of object-oriented interaction models with all relevant information for reflecting and improving design decisions.

C. Architecture

Based on the defined requirements and the use of the object-oriented log metamodel, we develop the architecture for our model-driven reverse engineering approach. Fig. 2 shows the architecture and describes the interplay of design time and runtime artefacts. The architecture can be divided into three parts: First, the creation of object-oriented interaction models in form of sequence diagrams from executed operations (Fig. 2, Circle 1); second, the alignment between these sequence diagrams and their corresponding design models for creating so-called trace links (Fig. 2, Circle 2); and third, the creation of a runtime profile and display of that profile in the design models (Fig. 2, Circle 3).

For the creation of sequence diagrams from system logs (Fig. 2, Circle 1), we need the executed operations at runtime transformed to object-oriented logs as described in the metamodel shown in Fig. 1. There are two options to create such logs. On the one hand, such log files can be created manually, based on the implementation (i.e., coding). On the other hand, it is feasible that such a specific logging is already considered at design time by annotating a defined stereotype to certain model elements as presented and evaluated in previous work [3]. For this purpose we extended a code generator that recognizes certain annotated model elements. For these, code is generated that automatically logs changes at runtime. Thereby we use a Class Diagram (CD) to describe the structure of the system and its properties and operations, and a State Machine (SM) to model the behavior by states and transitions (see Fig. 2, System@DesignTime, and Section II).

In a further step, the system log files (no matter if generated manually or automatically by the extended code generator) are used as input for a Text-to-Model (T2M) transformation for automatically deriving an object-oriented interaction model in form of a Sequence Diagram (SD) (cf. Section III-D).

Based on the metamodel described by Fig. 1, we perform an alignment between runtime and design time models (Fig. 2, Circle 2). Based on the defined classes and operations during design time we analyze the lifelines and messages to generate trace links between corresponding elements. This enables us to consider runtime and design time together. We query the elements of the SD in combination with the original CD and SM by an Application Programming Interface (API) (cf. Section III-D).

Based on these trace links we can go a step further by extracting some additional runtime information from the SD. This information is displayed as profile over runtime in the design model (Fig. 2, Circle 3). For this purpose we query our traces by the API to analyze if the message exchange works as intended and to count, e.g., the frequency of operation calls. This obtained runtime information is then saved as tagged values by a stereotype in the original design model (cf. Section III-D). This means that we learn from the runtime (actual) in order to constantly improve design models (target), as intended by the PDCA-cycle. But most importantly this reverse engineering approach enables an end-to-end traceability from design to runtime and back again. The elements can be examined on different levels, which enables to navigate from instance level to type level or the other way around.

D. Prototypical Implementation

For a prototypical implementation in a first version, we employ the modeling analysis and design tool Enterprise Architect\(^3\) (EA). By using this tool, we model the CD and SM and use the extended code generator, which we presented in [3], to automatically generate execution code from these design models. During operation the execution is recorded in form of object-oriented system logs as presented in the metamodel (see Fig. 1). For generating the interaction model as SD, we developed the EA Add-In “EA Sequence Miner”. It is written in C# with the EA Automation API ActiveX COM library\(^4\). This Add-In enables to automatically transform

\(^3\)https://www.sparxsystems.de/uml/neweditions/
\(^4\)http://www.sparxsystems.com/enterprise_architect_user_guide/13.5/
system logs into a SD by a T2M-transformation.

The generating process consists of the following steps:

1) A new SD is created.
2) A new lifeline is created for each of the different values of Sender and Receiver.
3) For every Message entry in the log file a message with the corresponding name, parameter and message type is created in the same time order as in the log trace from sender lifeline to receiver lifeline.
4) Saving the generated SD elements in the integrated database of the EA file.

In our prototypical implementation the system log is stored in a csv-File with the following structure:

<table>
<thead>
<tr>
<th>case</th>
<th>timestamp</th>
<th>Sender</th>
<th>Receiver</th>
</tr>
</thead>
<tbody>
<tr>
<td>2019-02-27 17:38:13.991</td>
<td>Car</td>
<td>Car</td>
<td></td>
</tr>
<tr>
<td>2019-02-27 17:38:13.992</td>
<td>Car</td>
<td>ServoControl</td>
<td></td>
</tr>
</tbody>
</table>

Such system log is used as input for the EA Sequence Miner to generate appropriate SD. Fig. 5 (not bold parts) shows an excerpt of an automatically generated SD of our motivating example of a self-driving car (cf. Section II). The SD shows the different instances of the car components, their lifelines, and the exchange of messages among them. Thus, the workflow of the car in a certain setting is easier to track and understand than per mining a text-based file.

For the alignment of runtime and design time information as well as for analysis purpose, we have developed an analysis tool in C# with the UniqueMint API\(^1\) developed by LieberLieber Software GmbH\(^2\) for EA. By using this library we are able to navigate through and to query UML models. After loading the generated SD by the UniqueMint API, we query the SD by using Language Integrated Query\(^3\) (LINQ). For example, to get the instance classifiers \((\text{represents})\) of our lifelines we checked the class names and compare them with the lifelines:

```csharp
public void SettingRepresentLifeline(IMyLifeline lifetime)
{
    lifetime.Represents = FindClass(lifeline.Name);
}
private IMyClass FindClass(String lifeline)
{
    var classProp = ClassList.AsParallel();
    Where(c => c.Name.Contains(lifeline).FirstOrDefault();
    return classProp;
}
```

The same principle is used for identifying if there exist belonging operations for the messages or not. In addition, parameters have to be considered. Return messages of calls are not linked to the original models. In Fig. 3 the established traces are shown in bold.

Based on these established trace links we now extend our CD by a runtime profile consisting of a stereotype for classes and operations. Fig. 4 shows the RuntimeProfile with the RuntimeMetaInfo stereotype and its tagged values. On the one hand, we store the information whether an operation or class is used during execution \((\text{isUsed})\) as boolean value, and on the other hand, we calculate how often instances or operations are used in the interaction \((\text{frequency})\) as integer value. For this purpose we use the already established trace links and again the UniqueMint API. Based on our LINQ queries, we extract the runtime information and saved it as tagged values for the specific operations and classes in the CD.

For instance to get frequency of a specific operation \((\text{operation})\) of a specific instance of a class \((\text{class})\) in the SD the following query searches through all messages \((\text{MessageList})\). According to the UniqueMint API \(m.\text{Signature}\) specifies the corresponding operation.

```csharp
List<IMyMessage> specificMessages = MessageList.AsParallel();
Where(m => m.Signature == "operation");
ToList();
int frequency = specificMessages.Count;
```

The full implementation of our approach can be found at

---

1. [https://demo.lieberlieber.com/EnArWeb/enarhtmlexport/MyUml/index.html](https://demo.lieberlieber.com/EnArWeb/enarhtmlexport/MyUml/index.html)
2. [https://www.lieberlieber.com/](https://www.lieberlieber.com/)
our project website\textsuperscript{8}.

IV. EVALUATION

In this section, we present and discuss the accuracy of our approach using a case study of a self-driving car based on the motivating example (see Section II). We follow the guidelines for conducting empirical explanatory case studies by Roneson and Höst [4].

A. Research Questions

Our general evaluation interest is the scalability of our presented approach. Therefore, in our study we aim to answer the following research questions (RQs):

\begin{itemize}
  \item \textbf{RQ1—Scalability of interaction model generation:} How long does it take to generate an interaction model from a system log?
  \item \textbf{RQ2—Scalability of interaction diagram generation:} How long does it take to generate a diagram for an interaction model? Which system log sizes can be displayed as sequence diagrams?
  \item \textbf{RQ3—Scalability of interaction model integration:} How long does it take to integrate the interaction model, i.e., runtime model, with the design model?
  \item \textbf{RQ4—Scalability of interaction profiling:} How long does it take to compute the profiling information?
  \item \textbf{RQ5—Overall performance:} How long does it take in total to (a) show a interaction model to an engineer, and (b) to present the profiling information to an engineer?
\end{itemize}

B. Case Study Design

\begin{itemize}
  \item \textbf{a) Requirements:} As an appropriate input for our case study, we require an automated system such as a self-driving car where at runtime system logs can be observed as object-oriented logs. In addition, we require that the structure of the system is modeled with UML class diagrams (CD) and instances of the system logs can be uniquely assigned to classes of the CD.
  \item \textbf{b) Setup:} For our evaluation, we use different system runs resulting in different object-oriented system logs regarding the number of messages between the individual components of the self-driving car (see Table I). In our self-driving car case with four stationary components, the number of messages therefore varies from 10 to 100,000. The upper limit can be explained by the fact that with 100,000 messages, a broad spectrum of different interaction patterns can occur and thus can be analyzed by our approach.
\end{itemize}

\begin{table}[h]
\centering
\begin{tabular}{|c|c|c|c|c|c|c|c|}
\hline
System Logs & 10 & 100 & 1,000 & 10,000 & 50,000 & 100,000 \\
\hline
\end{tabular}
\caption{Different sizes of the investigated system logs}
\end{table}

For answering our RQs, we calculate the duration between the start of SD generation and the finished process and the duration for each query by System.Date\text{Time} in C\# in milliseconds (ms). The performance is measured on an Acer Aspire VN7-791 with an Intel(R) Core(TM) i7-4720 HQ CPU@2.60 GHz 2.60 GHz, with 16 GB of physical memory, and running Windows 8.1. 64 bits operating system. Please note that we measured the CPU time by executing each query and SD generation three times for all different settings and calculated the arithmetic mean of these runs in milliseconds (ms). To create our models we use Enterprise Architect version 13 with the integrated Microsoft Access database to store the models.

C. Results

We now present the results of applying our approach to the different settings of our self-driving car. Table II shows the execution times for generating the interaction models without visualization in a diagram (only saving the components in the database) and with visualization. In addition, the required disk space of the persisted models in the database is shown. It is noticeable that the difference between generation with and without visualization is marginal. However, the duration of the generation of the diagram rises sharply from a size of 50,000 messages. Fig. 6 (Interaction Model with and without Visualization) shows that the process is not linear, but polynomial (2nd degree). On the other hand, database sizes behave linearly as shown in Fig. 5.

For analysis of the creation of trace links and runtime profiles, we assume that the models are already loaded in memory. Therefore, we only analyze the concrete duration for the execution of the different queries. Table III shows the execution times for establishing the trace links and generating the runtime profile.

As before for the generation of sequence diagrams, the distribution of the data for establishing trace links and for generation the profile correspond in each case to a quadratic function.

Based on these execution times it is now possible to calculate average of the overall performance of the approach (see Table IV). It is significant that the process of generating and loading the model in memory is the most consuming task of the approach. The influence of generating profiling

\vspace{1cm}

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{fig5.png}
\caption{Database size in relation to number of messages in the SD (in MB)}
\end{figure}
TABLE II
DATABASE SIZES AND EXECUTION TIMES FOR GENERATING INTERACTION MODELS

<table>
<thead>
<tr>
<th>Database Generation without Visualization (s)</th>
<th>Database Generation with Visualization (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>#Messages (MB) Run 1 Run 2 Run 3 Average Run 1 Run 2 Run 3 Average</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>1.61 0.44 0.44 0.44 0.44 0.44 0.44 0.44 0.44 0.44 0.44</td>
</tr>
<tr>
<td>100</td>
<td>1.66 3.44 3.44 3.44 3.44 3.44 3.44 3.44 3.44 3.44 3.44</td>
</tr>
<tr>
<td>1,000</td>
<td>2.11 3.63 3.63 3.63 3.63 3.63 3.63 3.63 3.63 3.63 3.63</td>
</tr>
<tr>
<td>6,000</td>
<td>4.94 245.15 246.03 246.42 245.87 245.31 245.31 245.31 245.31 245.31 245.31</td>
</tr>
<tr>
<td>10,000</td>
<td>7.11 454.14 461.04 459.25 458.31 452.99 453.44 453.44 453.44 453.44 453.44</td>
</tr>
<tr>
<td>30,000</td>
<td>26.90 1,002.54 1,003.92 1,003.92 1,003.92 1,003.92 1,003.92 1,003.92 1,003.92 1,003.92 1,003.92</td>
</tr>
<tr>
<td>90,000</td>
<td>57.70 2,473.66 2,466.72 2,469.46 2,469.79 2,468.25 2,473.28 2,473.28 2,473.28 2,473.28 2,473.28</td>
</tr>
</tbody>
</table>

TABLE III
EXECUTION TIMES FOR TRACE LINKS AND GENERATING RUNTIME PROFILE

<table>
<thead>
<tr>
<th>Trace Links (s)</th>
<th>Runtime Profile (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>#Messages</td>
<td>Run 1 Run 2 Run 3 Average</td>
</tr>
<tr>
<td>10</td>
<td>0.016 0.016 0.016 0.016 0.016 0.024 0.025 0.025 0.025</td>
</tr>
<tr>
<td>100</td>
<td>0.016 0.016 0.016 0.016 0.024 0.025 0.025 0.025 0.025</td>
</tr>
<tr>
<td>1,000</td>
<td>0.016 0.016 0.016 0.016 0.021 0.021 0.021 0.021 0.021</td>
</tr>
<tr>
<td>6,000</td>
<td>0.024 0.024 0.024 0.024 0.024 0.024 0.024 0.024 0.024</td>
</tr>
<tr>
<td>10,000</td>
<td>0.024 0.024 0.024 0.024 0.024 0.024 0.024 0.024 0.024</td>
</tr>
<tr>
<td>50,000</td>
<td>0.024 0.024 0.024 0.024 0.024 0.024 0.024 0.024 0.024</td>
</tr>
<tr>
<td>100,000</td>
<td>0.024 0.024 0.024 0.024 0.024 0.024 0.024 0.024 0.024</td>
</tr>
</tbody>
</table>

TABLE IV
AVERAGE EXECUTION TIME OF THE OVERALL PERFORMANCE OF THE APPROACH

<table>
<thead>
<tr>
<th>#Messages</th>
<th>Interaction Model SD (s)</th>
<th>Profiling Information (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>0.016</td>
<td>0.030</td>
</tr>
<tr>
<td>100</td>
<td>0.016</td>
<td>0.030</td>
</tr>
<tr>
<td>1,000</td>
<td>0.016</td>
<td>0.030</td>
</tr>
<tr>
<td>6,000</td>
<td>0.024</td>
<td>0.024</td>
</tr>
<tr>
<td>10,000</td>
<td>0.024</td>
<td>0.024</td>
</tr>
<tr>
<td>50,000</td>
<td>0.024</td>
<td>0.024</td>
</tr>
<tr>
<td>100,000</td>
<td>0.024</td>
<td>0.024</td>
</tr>
</tbody>
</table>

Answering RQ1: Our investigations of the execution time for generation interaction models from system logs already show for this use case the influence of number of messages in the system log. At a certain size (around 50,000 messages), the execution time no longer increases linearly. This means for larger system logs, a different storage technique may have to be found. The main reason for that seems to be that the used API writes each record individually in the database whereas bulk operations may be more applicable here. However, further investigations to substantiate this statement are needed in the future.

Answering RQ2: The effort to generate the graphical visualization from interaction models is very low in the generation process. The main effort consists in creating the specific elements in the database. The SD could be created for all investigated test cases.

Answering RQ3: Once the model has been loaded into the memory, the trace links are set up quickly. However, from 50,000 messages onwards, the analysis and assignments slows down and the execution time increases according to a quadratic function. However, the trace links could still be established for all settings.

Answering RQ4: The calculation of the profile information takes as long as the establishment of the trace links regarding a number of messages up to 6,000. With a larger number, the distribution also increases in a polynomial way, but less strongly.

Answering RQ5: The overall execution shows that the generation of the models takes the most time. Once the models are loaded into memory, optimized queries can be performed. However, the influence of the number of messages should not be underestimated as the evaluation of the execution times.
shows.
Our evaluation shows that scaling of modeling tools in MDE is still a crucial issue as described in [5].

D. Threats to Validity

Internal Validity - Are there factors that can influence the results of the case study? In our evaluation we only vary the number of messages, the scaling of the lifelines is not examined. Also, our evaluation is limited to a maximum of 100,000 messages. For execution with a larger number of messages, further studies are needed.

External Validity - Is it possible to generalize the results? Our investigations are limited to the type of use case described in the setup of the case study. In order to generalize the approach, the evaluation should be carried out with other use cases by persons without knowledge of the internal realization of the presented approach.

V. RELATED WORK

In this section, we present and discuss related research works.

Process mining (PM) serves as a bridge between design time and runtime, by combining data mining and business process modeling to a new research field [6]. PM enables to extract process-related information from so-called event logs [6], [7]. PM defines events as process steps and event logs as sequential ordered events recorded by any information system during operation [8]. Such actual workflow information is then used to discover process models for enabling a target/actual comparison, e.g., to identify bottlenecks [6]. PM is also applied in software engineering. Ladiges et al. [9] show an approach where they demonstrate a learning algorithm for automatic generation of material flow Petri nets based on recorded PLC I/O data. Their approach allows a documentation of the processes, where anomalies in the material flow are detected and process properties can be determined. In [10], the authors present a novel reverse engineering technique to obtain real-life event logs from distributed software systems. Thereby, PM techniques are applied for obtaining precise and formal models, and to monitor and improve processes by performance analysis as well as conformance checking.

Similar to PM, our approach aims for visualizing runtime information as interaction model in a sequence diagram. However, we go a step further and use the provided information to automatically identify sequence patterns in the model to get a deeper insight about interaction frequencies, occurrences of properties and to back propagate this information in the system design model.

The research field of Reverse Engineering (RE) deals with reversing information from runtime to design time. This means, RE takes running code and elevates it on a higher level of abstraction (e.g., model level) for analyzing it. Raibulet et al. [11] conduct a systematic literature review on RE in the application field of MDE. Thereby, the authors analyze different used model languages, transformations, as well as the degree of automation the used tools. The authors come up that most of the investigated approaches deal with source code such as Java or C or web application code to get a representation on object-oriented model level. In contrast, we analyze runtime logs of a system in order to annotate an existing design model.

In [12], the authors analyze the requirements and conditions that allow partial automation of model-based reverse engineering. In their experiments, the authors use the power of combining metaprogramming and metamodeling techniques. Our approach is different in so far as we are back propagating metainformation from runtime information into design models instead of only creating a runtime model. In [13], the authors analyze the difficulty of automatically finding errors in object-oriented code. For this purpose they use unit tests to get traces of execution paths. From this they create then sequence diagrams. In order to minimize the size of the diagrams, the authors aggregate between error-free and error-prone runs in order to find the incorrect code line. Their approach serves to find faults in a runtime code by trying to extract information from runtime for improving the design. This means that the authors perform a comparison and then display faulty parts. In our approach, we generate additional information from runtime to improve the design by annotating this metainformation to the design model.

Briand et al. [14] propose the generation of sequence diagrams for specific use cases by performing dynamic analysis to help people understanding system behavior. The authors formally describe the RE-process by using metamodels and transformation rules in OCL (Object Constraint Language). In contrast to our approach, the authors are focusing on executed source code instead of observed system logs. In [15], the authors use static information from source code and dynamic information from executing the code. Both information are then represented as models. Based on these models they develop a model transformation to generate a sequence diagram out of the code. In our approach, we automatically generated runtime models from executed system logs and automatically generate the corresponding sequence diagram from those logs, e.g., for validation purposes.

Younis and Frey [16] show an approach where the generate an UML Class Diagram and State Diagram out of PLC code. Their approach deals with the reverse engineering of detailed design information of the PLC code that was not previously available. Davydova and Shershakov [17] describe an approach to generate sequence diagrams from execution traces of SOA information systems. In contrast to the other approaches and similar to ours, they do not use executed source code for generating sequence diagrams, but using event logs like in PM. However, they do not further examine automatically generated sequence diagrams in order to annotate design models.

As discussed, there are similar approaches but they are focusing either on executed source code or on creating an interaction model from it. Our approach aims for analyzing logs from an operating system (e.g., controller of a machine), generating sequence diagrams and annotating properties in form of metainformation to existing design models.
VI. CONCLUSION AND FUTURE WORK

In this paper we presented an approach for automatically reverse engineering of interaction models from system logs and back propagation of runtime information into design models. Our case study about a self-driving car shows that the approach is feasible for annotating and tracing from runtime to design time. We are able to generate interaction models from system logs and then based on our developed queries, the relationships between runtime models and design models are established. Our evaluation shows that scaling of modeling tools is still a crucial issue. The strength of our approach is that we can keep the relevant information in a unified modeling language, namely UML. Thus, design tools can be reused with their integrated tooling and there is no need to learn new technologies or languages for analyzing runtime information. At the same time, the approach is depending on existing UML technologies, and thus, it may be challenging to use the approach in Big Data settings without any pre-processing. For such settings, the logs have to be decomposed before analysis.

Regarding our presented model-driven approach, we foresee the following next steps. First of all, we plan to vary the number of used lifelines by applying and validating our approach in another case studies such as distributed production systems, where multiple systems with different object-oriented system logs exist. We want to check the generalization of our approach. For instance, if IDs are for example hash values, then methods such as heuristic matching will have to be applied additionally. Second, we strive to adapt our approach to use the timing aspect not only to get the right order in the SD, but also to explicitly store how long messages take time. This allows, for example, to annotate information about minimum, maximum, average durations as tagged values of the stereotype for operations in the design model. Third, we plan to extend our approach to use the generated SD as input for the simulation of the SM in order to experiment even better with runtime data. For instance, in this simulation break points could be set to analyze certain processes more precisely or to manually trigger events and observe system reactions. Finally, we want to adapt the presented approach on the technology level to reach our main goal of linear scaling.

ACKNOWLEDGMENT

This work has been supported by the Austrian Federal Ministry for Digital and Economic Affairs and the National Foundation for Research, Technology and Development (CDG). We also thank Maximilian Medetz and Horst Kargl for their work and fruitful support.

REFERENCES

14 Model-driven Runtime State Identification

S. Wolny, A. Mazak, M. Wimmer and C. Huemer;
Corpus ID: 220055432
Model-driven Runtime State Identification

Sabine Wolny, Alexandra Mazak, Manuel Wimmer, Christian Huemer

Abstract: With new advances such as Cyber-Physical Systems (CPS) and Internet of Things (IoT), more and more discrete software systems interact with continuous physical systems. State machines are a classical approach to specify the intended behavior of discrete systems during development. However, the actual realized behavior may deviate from those specified models due to environmental impacts, or measurement inaccuracies. Accordingly, data gathered at runtime should be validated against the specified model. A first step in this direction is to identify the individual system states of each execution of a system at runtime. This is a particular challenge for continuous systems where system states may be only identified by listening to sensor value streams. A further challenge is to raise these raw value streams on a model level for checking purposes. To tackle these challenges, we introduce a model-driven runtime state identification approach. In particular, we automatically derive corresponding time-series database queries from state machines in order to identify system runtime states based on the sensor value streams of running systems. We demonstrate our approach for a subset of SysML and evaluate it based on a case study of a simulated environment of a five-axes grip-arm robot within a working station.

Keywords: Model-driven Engineering; Time-Series Database; State Identification; Runtime Queries; Process Mining

1 Introduction

Forecasts show that in the upcoming years most of the devices we interact with will be linked to a global computing infrastructure [BS14]. This tendency represents an infrastructure in which the physical environment is populated by interconnected and communicating objects (e.g., sensors, actuators and other smart devices) capable for autonomously interacting with each other and with the environment itself. In order to deal with the increasing complexity of cyber-physical systems (CPS), models are used in many research fields as abstract descriptions of reality. This means that a model serves as an abstraction for a specific purpose, as a kind of “blueprint” of a system, describing the system’s structure as well as desired behavior. However, often we recognize a discrepancy between these models and their real world correspondents [MW16b]. In other words, we experience deviations between design-time models and runtime models discovered from real data.
This development raises new challenges for Model-Driven Engineering (MDE) approaches [MWP18]. While design models help in the engineering process by providing appropriate abstractions, data-driven approaches such as process mining [Aa16] may help to uncover some under-specified or unintended parts of these design models at runtime. For instance, on a high level of abstraction, behavioral modeling languages (e.g., state-machine-based languages) are used to describe the behavior of a physical asset by means of states and transitions. Such models define discrete states, which are represented by defined variable values. A system has to achieve/go through these states during its execution. However in reality, systems do not switch in a time discrete manner between states, but the values of the variables are continuously evolving to the intended values of the next state. This means, each variable undergoes a continuous series of changes that need to be continuously monitored, e.g., to be able to react immediately to a time delay in safety critical systems. The challenge is to continuously listening to value streams in order to determine whether a state has indeed occurred, i.e., if the specific combinations of variable values have occurred over all streams at the same time. In particular, the realization precision of systems as well as measuring inaccuracies complicate this process as false positives and false negatives may occur when matching state templates to data streams.

Based on first ideas presented in [Wo17], we address this challenge by introducing a novel approach where we automatically generate state realization event queries derived from state machines for an appropriate state identification at runtime. This approach enables us to continuously observe multiple data streams of distributed sensor devices for identifying a system’s entire state at runtime. By applying the so-called Model-driven Runtime State IdEntification (MD-RISE) approach, we automatically transform behavioral models, i.e., state machines, into time-series queries to be able to match sensor value streams with pre-defined variable values of the design model to report identified states from execution. First evaluation results derived from a case study of a 5-axes grip-arm robot show the potential of the approach in terms of precision and recall of finding system states in sensor value streams. By this, state based monitoring is possible for instance, even if the systems are not able to provide a explicit state-based trace.

The remainder of this paper is structured as follows. In the next section, we present a motivating example for this paper. Section 3 presents the MD-RISE approach by describing the MD-RISE architecture and its prototypical implementation. Section 4 demonstrates the evaluation of MD-RISE based on a case study of a 5-axes grip-arm robot which is interacting with other components within a working station, like a pick-and-place unit. In Section 5, we discuss related work. Finally, we conclude this paper by an outlook on our next steps in Section 6.

2 Motivating Example

As motivating example for this paper, we consider a simple continuous automated system around a 3-axes grip-arm robot (gripper). This gripper is modeled by using by the Systems
Modeling Language (SysML) [FMS12], in particular by using the block definition diagram (BDD) and the state machine diagram (SM). The BDD is used to define the structure of the gripper with its properties: BasePosition (BP), MainArmPosition (MAP), and GripperPosition (GP) (see Fig. 1(a) Design Models, BDD System). These properties describe the angle positions of the three axes of the gripper. Based on the machine operator’s knowledge, these angle positions can be defined for different settings (e.g., drive down, pick-up) with pre-defined tolerance ranges. These ranges fix the accepted margin of deviation (e.g., ±0.1) for the variable values of each property (BP, MAP, GP). The desired behavior of the gripper is described by various states and state transitions modeled by using the SM (see Fig. 1(a) Design Models, SM Grip-arm robot). These states are drive down and pick up with assigned variable values specifying the respective angle position in these states.

During operation (i.e., execution at runtime), the gripper as a continuous system moves in its environment (e.g., pick-and-place unit) on the basis of a workflow described by the SM. These movements are recorded by various axis sensors and returned as continuous sensor value streams on a log recording system. In our motivating example, we record three sensor value streams BP, MAP, GP (see Fig. 1(b) Runtime Data). These records show that the gripper does not “jump” from one discrete state into another as modeled in the SM, but is-of course—continuously moving. Thus, the challenge is to identify possible discrete states by analyzing the sensor value streams. For this purpose we have to raise raw sensor value streams on a higher level of abstraction. This enables, e.g., to better compare an initial model (e.g., SM) with its realization.

The state identification is done by matching the different raw sensor value streams to the pre-defined variable values defined in the SM (see Fig. 1(b) Runtime Data). It should be...
considered that the pre-defined absolute variable values in the SM are not necessarily precisely measured in the real world because of, e.g., measuring inaccuracies. Such inaccuracies has to be taken into account by dealing with numerical values of objects of the physical world [MWV16]. Thus, in order to perform the state identification successfully, it is important to define appropriate tolerance ranges (see Section 4). For instance, the sequence of identified states can be used as input for further analysis (see Section 3).

3 Model-driven Runtime State Identification

In this section, we present our Model-driven Runtime State IdEntification (MD-RISE) approach which combines MDE-techniques with a Time-Series Database (TSDB) and Process Mining (PM), for states identification, recording, abstraction, and analyses. Fig. 2 shows the architecture of MD-RISE as well as the interplay of design time and runtime artefacts.

3.1 MD-RISE Prerequisites

For prototypically realizing the approach, we have a number of prerequisites that must be met: (i) the system’s workflow must be expressible by means of a state machine, (ii) the different states of the system must be unique in order that values describing a state are not identical for two different states, (iii) numeric values must be returned by sensors at runtime and must be storable in a TSDB, and (iv) it must be ensured that the time stamps are accessible.
3.2 MD-RISE Architecture

Based on the motivating example of the gripper (see Section 2) and the mentioned prerequisites, we consider an automated system consisting of a controller, sensors, and actuators. At design time, we model the structure and behavior of this system by using a subset of SysML (see Figure 2: System@DesignTime, BDD and SM). Fig. 3 shows the simplified graphical metamodel used for modeling BDD and SM of the system. Every component of the system (Block) contains properties (Property) and can have a SM (StateMachine), which describes the behavior of this component. Each Property can have a specified tolerance range (ToleranceRange) that defines an acceptable deviation of the assigned property values, e.g., based on measurement inaccuracies. The SM consists of states (State) and transitions (Transition). Generally, a state can have multiple incoming and outgoing transitions. A transition must have a predecessor and successor state (see Fig. 3). Additionally, different values can be assigned to a state (Assignment). In this paper, we just focus on Float property values, since we are interested in value changes during execution (see Section 4).

Based on this metamodel, we automatically derive a query on the basis of the SM, a so-called “state realization event query” (see Fig. 2, System@RunTime). This query helps for identifying states based on the recorded sensor value streams in a TSDB. For this purpose we use a Model-to-Text (M2T) transformation to automatically transform model elements to query statements (i.e., text strings) (see Subsection 3.3). During runtime, the sensors of the running system continuously send data over a messaging system middleware. These sensor value streams (e.g., values of the angle positions of the gripper) are recorded in a TSDB (see Figure 2). A single log of the stream contains the following information: timestamp (the actual time in the granularity of seconds), sensor (the name of the specific sensor), value (the measured value). The number of log entries for one component varies depending on the number of sensors. The challenge is to continuously listening to value streams in order to determine whether a state has indeed occurred, i.e., if the specific combinations of variable values have occurred over all streams at the same time (see Fig. 1(b) Runtime Data). For this purpose we apply the aforementioned state realization event query for identifying
states containing the following information: timestamp (the actual time in the granularity of seconds), state (the recognized state based on measured values).

However, the absolute values assigned in the SM at design time (see Fig. 1(a) Design Models) are necessarily not precisely identified as such during runtime due to measurement inaccuracies. For instance, we define for a certain state (e.g., driveDown) a value of 1.50 for a certain angle position (e.g., MAP) at design time, but at runtime we measure a value for this position of 1.492. For this purpose we implement a tolerance range, assigned to the initial model (e.g., the SM), to define in which range such inaccuracies are still acceptable (see Figure 3, ToleranceRange). The definition of such a range is crucial. If the range is selected too small, the inaccuracies may result in too few or even no identified states. Otherwise, if the range is too large, too many states are identified. We examine this challenge in our case study presented in Section 4.

In a next step, we generate a state-based log model that consists of the information of all identified states and, in addition, a case ID for identifying the corresponding process instance (see Fig. 2: System@RunTime, State-based Log Model). Such a case ID is required when using PM tools in order to be able to distinguish different executions of the same process. We employ this case ID in our approach to identify single runs of the SM during runtime.

In a further step, the state-based log model is transformed to an event-based log model (see Fig. 2, Event-based Log Model) by applying a Model-to-Model (M2M) transformation, like presented in previous research work [MW16a]. Since, we use a PM tool for analyzing this model, the structure must be based on eXtensible Event Stream (XES) schema. This is a supported input format of ProM Lite 1.1. For instance, by using this PM tool, the event-based log model can be analyzed, e.g., to uncover some under-specified or unintended events that were not considered in the SM.

In summary, by applying MD-RISE it is now possible to raise raw sensor value streams on a higher level of abstraction, namely the state level. MD-RISE bases on queries, so-called state realization event queries, which are automatically derived from an initial design model for the purpose of state identification at runtime. The identified system states can be automatically transformed into a state-based log model to make the outcome useable, e.g., for PM tools like ProM Lite for further analyses.

3.3 MD-RISE Prototypical Realization

For a first prototypical realization, we use the defined metamodel (see Fig. 3) and implement it by using Ecore in the Eclipse Modeling Framework. Based on this metamodel, we develop a M2T transformation by using Xtend in order to automatically generate state realization event queries out of the SM for different states. The structure of this M2T
transformation depends on the used TSDB. In our implementation, we use InfluxDB as TSDB. Therefore, the structure of our state realization event queries are similar to a SQL syntax, as shown in the following pseudo code example based on our metamodel:

```sql
FOR s IN Block.stateMachine.state
SELECT FOR a IN s.assignment "a.property.name", "ENDFOR" time
FROM "Block.name"
WHERE FOR a IN s.assignment
"a.property.name">="a.value-a.property.tolerance.value"
and "a.property.name"<="a.value-a.property.tolerance.value""ENDFOR"
"ENDFOR"
```

Based on the raw sensor value streams collected at runtime and stored in the TSDB, the queries are executed and the results are the identified states with their timestamps. In our prototypical implementation, we store the outcome as csv-file, which is then used as input for the state-based log model. This model is a Ecore model representation of the csv-file. In a next step, we use the Atlas Transformation Language (ATL) as transformation tool to transform the state-based log model to an event-based log model for importing it into ProM [MW16a]. The full implementation of MD-RISE can be found at our project website.

4 Case Study based on a CPPS-Simulation Environment

In this section, we present as well as discuss the accuracy and limitations of MDE-RISE on the basis of a case study of a CPPS-simulation environment around a 5-axes grip-arm. In doing so, we follow the guidelines for conducting empirical explanatory case studies by Roneson and Hörst [RH09]. In particular, we report on applying our approach to detect states at runtime based on stored value streams in a TSDB.

4.1 Research Questions

The study was performed to quantitatively assess the completeness, correctness, and performance of MDE-RISE. More specifically, we aimed to answer the following research questions (RQs):

- [https://www.influxdata.com](https://www.influxdata.com)
- [https://www.eclipse.org/atl](https://www.eclipse.org/atl)
- [http://promtools.org/doku.php](http://promtools.org/doku.php)
RQ1—Correctness: Are the identified states at runtime correct in the sense that all identified states are representing real states? If our approach identifies incorrect states, what is the reason for this?

RQ2—Completeness: Are the identified states complete in the sense that all expected states are correctly identified? If the set of identified states is incomplete, what is the reason for missed identifications?

RQ3—Performance: How strongly is the performance of the query execution influenced by the number of sensor value streams or the number of stored values per sensor?

4.2 Case Study Design

Requirements. As an appropriate input for our case study, we require an automated system such as a gripper integrated in a simulated environment where we are able to observe the behavior of the gripper during operation. We require access to multiple sensors of the gripper for log acquisition and a method to automatically identify states based on sensor value streams from simulation runs.

Setup. To fulfill these requirements, we implemented a CPPS-simulation of an autonomous acting production unit executed by using the open source tool Blender. The simulation scenario considers a working station, like a pick-and-place unit, where a gripper takes work pieces from a conveyor belt, puts them down on a test rig, and finally releases them in a red or green storage box based on the information coded on each work piece by a QR-code. Each component communicates via a messaging system middleware with InfluxDB. This TSDB provides us to acquire raw sensor value streams. During simulation, the gripper enters several different states for processing the work pieces. To verify the correctness of our approach, we have chosen two very similar states (differ only in one sensor value stream) to determine if the detection works: DriveDown and PickUp. The assigned values of the axes Base Position (BP), Main Arm Position (MAP), Second Arm Position (SAP), Wrist Position (WP), and Gripper Position (GP) of the two states in the SM are shown in Tab. 1. Furthermore we need to define an acceptable tolerance range to determine when the state identification is as accurate and complete as possible. We use a tolerance range from a deviation of 0 to a deviation of 0.4 (in 0.01 steps). The upper bound is only set for evaluation purposes to show the distribution of precision and recall. In reality, a deviation of 0.4 may be already too large. The deviation values are added or subtracted to the respective SM values (see Tab. 1). We use the same tolerance ranges for all properties and do not vary them.

For our evaluation we use two different database settings in combination with different numbers of sensor value streams that are used for the states identification. We use a dataset

https://www.blender.org
Tab. 1: Expected values for the gripper’s axes for the states DriveDown and PickUp.

<table>
<thead>
<tr>
<th>Gripper Axis</th>
<th>DriveDown</th>
<th>PickUp</th>
</tr>
</thead>
<tbody>
<tr>
<td>Base Position (BP)</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>Main Arm Position (MAP)</td>
<td>1.50</td>
<td>1.50</td>
</tr>
<tr>
<td>Second Arm Position (SAP)</td>
<td>-0.12</td>
<td>-0.12</td>
</tr>
<tr>
<td>Wrist Position (WP)</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>Gripper Position (GP)</td>
<td>1.5</td>
<td>-0.40</td>
</tr>
</tbody>
</table>

with 156 rows and a dataset with 1,560 rows stored in the database. For the state identification we use a single sensor value stream (GP), three sensor value streams (GP, BP, MAP), and all five sensor value streams (GP, BP, MAP, SAP, WP). For the performance check we also extend our dataset up to 15,600, 156,000, and 1,560,000 rows.

For performance purpose of the state realization queries, we calculate the duration between start of the query execution and result return by System.nanoTime() in Java. The performance figures have been measured on an Acer Aspire VN7-791 with an Intel(R) Core(TM)i7-4720 HQ CPU @ 2.60 GHz with 16 GB of physical memory, and running the Windows 8.1, 64 bits operating system. Please note that we measured the CPU time by executing each query 40 times for all different settings and calculated the arithmetic mean of these runs in milliseconds (ms).

Measures. In order to assess the accuracy of our approach, we calculate precision and recall as defined in [MRS08]. In the context of our case study, precision denotes the fraction of correctly identified states among the set of all identified states. Recall indicates the fraction of correctly identified states among the set of all actually occurring states. Precision denotes the probability that a identified state is correct and the recall is the probability that an actually occurring state is identified. Both values range from 0 to 1.

Precision is used to answer RQ1 and recall to answer RQ2. Furthermore, we calculate the so-called $f$-measure to avoid having only isolated views on precision and recall [MRS08]. To answer RQ3, we compute the duration of the query execution.

To check if our approach is accurate for a given scenario to identify system states, we have manually obtained the gold standard of state identifications for our given case study (156 rows: 3 expected states for DriveDown and PickUp, 1560 rows: 30 expected states for DriveDown and PickUp). For computing precision and recall, we extract the true-positive values (TPs), false-positive values (FPs) and false-negative values (FNs), with the help of the expected state identifications. From the TP, FP and FN values we then compute precision, recall and $f$-measure metrics as defined by Olson and Delen [OD08, p. 138].
Tab. 2: Precision, recall and f-measure for a single sensor value stream (GP). Bold line marks the best fit.

<table>
<thead>
<tr>
<th>DriveDown</th>
<th>PickUp</th>
</tr>
</thead>
<tbody>
<tr>
<td>tolerance range</td>
<td>precision</td>
</tr>
<tr>
<td>0</td>
<td>NaN</td>
</tr>
<tr>
<td>0.01</td>
<td>0.08</td>
</tr>
<tr>
<td>0.02</td>
<td>0.1</td>
</tr>
<tr>
<td>0.03-0.05</td>
<td>0.07</td>
</tr>
<tr>
<td>0.06-0.08</td>
<td>0.07</td>
</tr>
<tr>
<td>0.09-0.11</td>
<td>0.07</td>
</tr>
<tr>
<td>0.12-0.15</td>
<td>0.05</td>
</tr>
<tr>
<td>0.20-0.30</td>
<td>0.05</td>
</tr>
</tbody>
</table>

4.3 Results

We now present the results of applying our approach to the different settings of our gripper simulation. Tab. 2–Tab. 4 show the results for precision, recall and f-measure for the two different states in the different value stream settings. The values are valid for both database settings (156 rows, 1560 rows), since there were no differences with regard to precision, recall and f-measure. This can be explained by the fact that the queries are independent of the number of values in the database. As soon as the sensor value streams are in the accepted tolerance range, the state is returned.

Tab. 3: Precision, recall and f-measure for three sensor value streams (GP, BP, MAP). Bold line marks the best fit.

<table>
<thead>
<tr>
<th>DriveDown</th>
<th>PickUp</th>
</tr>
</thead>
<tbody>
<tr>
<td>tolerance range</td>
<td>precision</td>
</tr>
<tr>
<td>0</td>
<td>NaN</td>
</tr>
<tr>
<td>0.01</td>
<td>0.08</td>
</tr>
<tr>
<td>0.02-0.08</td>
<td>0.07</td>
</tr>
<tr>
<td>0.09-0.10</td>
<td>0.07</td>
</tr>
<tr>
<td>0.11-0.12</td>
<td>0.06</td>
</tr>
<tr>
<td>0.13-0.15</td>
<td>0.05</td>
</tr>
<tr>
<td>0.16-0.18</td>
<td>0.05</td>
</tr>
<tr>
<td>0.20-0.30</td>
<td>0.05</td>
</tr>
<tr>
<td>0.31-0.39</td>
<td>0.05</td>
</tr>
</tbody>
</table>

It is noticeable that the states identification fails and no states are found if the tolerance range is too small. The larger the range, the more false states are detected and the precision decreases as expected. In Tab. 2 for the state PickUp it could be recognized that the precision value is really small (highest value 0.08), because of wrong states identification based on a
single sensor value stream. This can be explained by the fact that the gripper moves during the simulation and opens and closes the gripper arm in various locations (e.g., conveyor, test rig). These states do not differ in the value of GP but have a different BP. Thus, this one axis GP is not enough to identify the state PickUp. Furthermore, it is interesting that the use of all gripper’s axes for state identification PickUp leads to a lower recall for the tolerance range 0.01 (see Tab. 4).

Tab. 4: Precision, recall and f-measure for five sensor value streams (GP, BP, MAP, SAP, WP). Bold line marks the best fit.

<table>
<thead>
<tr>
<th>tolerance range</th>
<th>DriveDown precision</th>
<th>DriveDown recall</th>
<th>DriveDown f-measure</th>
<th>PickUp precision</th>
<th>PickUp recall</th>
<th>PickUp f-measure</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>NaN</td>
<td>0</td>
<td>NaN</td>
<td>0</td>
<td>NaN</td>
<td>0</td>
</tr>
<tr>
<td>0.01</td>
<td>NaN</td>
<td>0</td>
<td>NaN</td>
<td>0</td>
<td>0.35</td>
<td>0.5</td>
</tr>
<tr>
<td>0.02-0.08</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0.09-0.10</td>
<td>0.75</td>
<td>1</td>
<td>0.86</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0.11-0.12</td>
<td>0.75</td>
<td>1</td>
<td>0.86</td>
<td>0.8</td>
<td>1</td>
<td>0.75</td>
</tr>
<tr>
<td>0.13-0.16</td>
<td>0.75</td>
<td>1</td>
<td>0.86</td>
<td>0.5</td>
<td>1</td>
<td>0.67</td>
</tr>
<tr>
<td>0.17-0.18</td>
<td>0.75</td>
<td>1</td>
<td>0.86</td>
<td>0.43</td>
<td>1</td>
<td>0.6</td>
</tr>
<tr>
<td>0.19</td>
<td>0.75</td>
<td>1</td>
<td>0.86</td>
<td>0.25</td>
<td>1</td>
<td>0.4</td>
</tr>
<tr>
<td>0.20-0.23</td>
<td>0.6</td>
<td>1</td>
<td>0.75</td>
<td>0.25</td>
<td>1</td>
<td>0.4</td>
</tr>
<tr>
<td>0.22-0.3</td>
<td>0.6</td>
<td>1</td>
<td>0.75</td>
<td>0.25</td>
<td>1</td>
<td>0.375</td>
</tr>
<tr>
<td>0.31-0.39</td>
<td>0.5</td>
<td>1</td>
<td>0.67</td>
<td>0.25</td>
<td>1</td>
<td>0.375</td>
</tr>
</tbody>
</table>

Figure 4 shows the results of our performance check. It could be determined that the number of sensor value streams and the number of rows in the database both have an influence on the execution time.

Interpretation of results. Answering RQ1: The recognition of correct states depends on the defined tolerance range and the differentiability of states. A precondition for our

![Fig. 4: Performance Results (average execution time in ms) according to sensor value streams and rows in the TSDB.](image-url)
approach is the uniqueness of states. However, if the various states differ only slightly, the number of sensor value streams used for states identification is relevant for correctness.

Answering RQ2: The selected tolerance range and the number of sensor value streams are also decisive for the completeness of the states identification. The more sensor value streams are used, the more important individual sensor values become for the identification. In addition, the completeness of the identified states is better the larger the selected tolerance range is. In our evaluation we quickly achieve a good completeness. As soon as this is reached, the tolerance range should not be further increased, otherwise the correctness of the identified states suffers.

Answering RQ3: Our investigations of the execution time already show in this simple setting the influence of the number of data records in the database and the used number of different sensor value streams. However, the performance seems still promising for large cases as we experience a linear increase of execution time for all tested settings.

4.4 Threats to Validity

Internal Validity - Are there factors that can influence the results of the case study? At design time, values for our axis positions are assumed on the basis of, e.g., calibration values. At runtime, the same exact values are not always measured, but with a certain fluctuation range. Thus, a certain tolerance range must be defined at design time in which the values are accepted. In our case, we knew exactly which values to expect and were therefore able to keep our tolerance range small. However, this might not work with other settings.

External Validity - Is it possible to generalize the results? Our approach is based on queries automatically created by state machines. We focus on creating queries that are understood by the TSDB InfluxDB. Thus, the queries are currently in SQL syntax. If a different database query language is needed, only the Xtend code has to be adapted regarding syntax without changing the model in the background. At the moment our evaluation is based on a single case of a gripper simulation. For further and more detailed results the study has to be extended to other scenarios. Raw data from sensors are often noisy, incomplete and can contain erroneous records. This is not considered in our case study. In addition, the datasets for performance analysis are relatively small in relation to databases. Larger sets would be needed for further more detailed results.

5 Related Work

Discovering the behavior of running software. In [Li16], the authors utilize process mining (PM) techniques to discover and analyze the real behavior of software. By doing so, they discover behavioral models for each software component by considering hierarchies. In a first step of their approach, they identify component instances and construct event logs
for each component from raw software execution data. In a second step, they recursively transform the logs to a hierarchical event log for each component by considering calling relations among method calls. Based on these hierarchical event logs, the authors discover a hierarchical process model to understand how the software is behaving at runtime. The authors' software component behavior discovery builds on the inter-disciplinary research field of Software Process Mining (SPM), firstly introduced by Rubin et al. [Ru07]. Both approaches base their grounding on the well-established techniques and methods of the research field of PM [Aa16].

Applying reverse engineering for obtaining event logs. In [LA15], the authors present a reverse engineering technique based on PM for obtaining real event logs from distributed systems. Similar to [Li16], the authors present an inter-disciplinary approach based on PM techniques and reverse engineering. The aim of their approach is to analyze the operational processes of software systems when running. The formal definition, implementation, and instrumentation strategy of the approach bases on a joinpoint-pointcut model (JPM) known from the area of aspect-oriented programming [EFB01]. This JPM helps (i) by defining the parts of a system that are to be included, (ii) enables to quickly gain insight into the end-to-end process, and (iii) detects the main bottlenecks. The authors demonstrate the feasibility of their approach by two case studies.

Query-based process analytics. A query approach enabling business intelligence through query-based process analytics is presented by Polyvyanyy et al. [Po17]. In contrast to our approach they are focusing on PM techniques for the automated management of model repositories of designed and executed processes, and on the relationships among these processes. For this purpose the authors introduce a framework for specifying generic functionalities that can be configured and specialized to address process querying problems, such as filtering or manipulation of observed processes.

Finally, we would like to highlight two research works that underline our approach and discuss the differences. Mayr et al. [Ma17] critically note that models are mainly used as prescriptive documents. Therefore, the authors aim for a model-centered architecture paradigm to keep models and developed artefacts synchronized in all phases of software development as well as in the running system. In this context, our approach helps to lift raw sensor data through automated states identification during operation at a model level for enabling a comparison between prescriptive and descriptive models. Senderovich et al. [Se16] apply PM techniques for real-time locating systems. They solve the problem of mapping sensor data to event logs based on process knowledge since location data recordings do not relate to the process directly. Therefore, they provide interactions as an intermediate knowledge layer between the sensor data and the event log [Se16]. Contrary to our approach, their raw sensor log consists already of different business entities and they have to map interactions to activity instances, while the sensor logs in our approach consist only of numerical values which we first have to aggregate to events.
6 Conclusion and Future Work

In this paper, we presented an approach that automatically derives state realization event queries from the design model to identify system states of a continuous system based on sensor value streams at runtime. This enables to raise raw sensor data from the data layer on a higher model layer. At this model level, runtime processes can be analysed more quickly and possible unintended parts within the realized system may be identified more easily and time-saving. Since inaccuracies has to be taken into account by dealing with numerical values of objects of the physical world, additionally we implemented a tolerance range for defining in which range such inaccuracies are still acceptable for an identified state at runtime.

First results of our case study indicate that a high precision and recall of system state identification may be achieved if an appropriate tolerance range for the runtime values was defined. Nevertheless, the uniqueness and distinctiveness of the individual states determine whether the state identification works well or not. If states are very similar, enough different sensor value streams must be used for state identification to obtain a good precision and recall. The approach is a step towards a better integration of model-driven software development to all the operations within a system’s life cycle in order to continuously deploy stable versions of application systems.

There are several lines for future work we are going to explore in more detail. First, we plan to apply and validate our approach in a real-world setting, instead of a simulation. Second, we want to extend our approach to monitor different components with a larger set of sensor value streams. Third, we only used identically tolerance ranges for the properties. In a further investigation, we want to find out if there are automated techniques possible to estimate good guesses for the tolerance ranges of different properties. Finally, we want to find out if we could extend our approach for state estimation and detection of possible hidden states.

Acknowledgment

This work has been supported by the Austrian Federal Ministry for Digital and Economic Affairs, the National Foundation for Research, Technology and Development, and by the FWF in the Project TETRABox under the grant number P28519-N31.

References

14 Model-driven Runtime State Identification


15 From AutomationML to AutomationQL: A By-Example Query Language for CPPS Engineering Models

M. Wimmer and A. Mazak;
DOI: 10.1109/COASE.2018.8560448
From AutomationML to AutomationQL: A By-Example Query Language for CPPS Engineering Models

Manuel Wimmer and Alexandra Mazak

Abstract—Model-based engineering is an emerging paradigm to deal with the complexity of multi-disciplinary engineering in CPPS projects. In such projects, different kinds of models are created during the lifecycle of a production system. AutomationML is a promising standard to provide a unifying format to represent and connect the different engineering models. Dedicated tool support has been developed for AutomationML in the last years to create and evolve models. However, when it comes to querying AutomationML models, implementation-related query languages have to be currently used. These languages have a certain complexity as they are not directly based on the concepts of AutomationML but on the underlying technological concepts and encodings of AutomationML. This often hinders the formulation of automatically executable queries by domain experts.

In this paper, we propose a dedicated query language for AutomationML called Automation Query Language (AutomationQL) which is directly derived from AutomationML. Using this query language, queries can be defined in a by-example manner which allows engineers to formulate queries in terms of AutomationML concepts instead of switching to an implementation-oriented query language. We illustrate how AutomationQL is defined, how queries can be formulated as well as how tool support is provided to automatically evaluate the queries and represent their results. Finally, we contrast our solution with existing query languages and derive a roadmap for future research on AutomationQL.

I. INTRODUCTION

Industrial production systems engineering is a multi-disciplinary process involving activities of different engineering disciplines starting with the overall system design on a production function level, moving to the detailed mechanical, electrical, and software engineering, and finally leading to the installation, commissioning, and operation of the production system [1], [2]. Faltinsky et al. [2] identify four main aspects of modern engineering processes: (i) tool-supported information exchange, (ii) model reuse and adaptability, (iii) executable models for early verification and validation, and (iv) a system-wide planning of the distributed system.

In support of such a seamless development process, the AutomationML (AML) standard [3] has been proposed and continuously developed.

AML is designed as a flexible language able to represent a large spectrum of different engineering artifacts as well as for harmonizing engineering data exchange within a heterogeneous tool network [4], [5]. AML is also utilized to build reusable libraries containing reusable hardware and software component types, such as sensors, vendor-specific production equipment, and PLC controllers, to build up production systems. Engineering artefacts which are uniformly represented as a collection of AML models may be analyzed with dedicated tool support such as employing query engines to select elements of interest in large and distributed models [6], [7].

We focus in this paper on the core part of AML [2], [3] which is the Computer Aided Engineering Exchange (CAEX) [8], [9], an IEC standard providing a neutral and open data format for the storage of hierarchical object information. Thus, CAEX is used to represent fundamental characteristics of a production system which should allow the exchange of CAEX-compliant artifacts throughout the complete system life cycle.

To further utilize the benefits offered by AML, we show in this paper how AML may be used as a basis for deriving an AML-specific query language called AutomationQL (AQL). In particular, we propose to merge a query language for finding graph patterns in AML models into the AML language. The resulting query language allows to formulate queries in AML structures instead of switching to specific technology-specific encodings of AML to formulate queries with technology-specific query languages. Furthermore, the resulting query language allows to formulate queries in a similar manner as existing by-example query approaches [10], [11]. This means, the query is defined by stating how the results should look like by giving an example instead of defining how they are computed. By following this by-example approach and reusing the AML syntax to formulate the queries as much as possible, domain experts with AML know-how are empowered to model queries in a similar way as they define AML models. For automatically executing the queries on AML models, we provide an interpreter. The query results computed by the interpreter are explicitly represented as proxy models which reference the found graph pattern occurrences to the base AML models. This allows to process query results in any direction, e.g., to generate a documentation of the query results or to provide transformations for the found graph pattern occurrences. AQL is implemented on basis of our Eclipse-based AML workbench which is publicly available [7].

The remainder of this paper is as follows: Section 2 briefly introduces AML and discusses existing query support. Section 3 explains our proposal by describing the definition of AQL and accompanying tool support. For a better understanding of the approach, several examples are presented in Section 4. Section 5 critically discusses our approach, and finally, Section 6 concludes and sketches future work.
II. BACKGROUND

In this section, we shortly introduce AML and discuss CAEX as we mainly focus on this part of AML. Furthermore, we discuss the current querying support for AML models as well as the general ideas behind query by-example approaches.

A. AutomationML

AutomationML (AML) [12] is a neutral XML-based data format for representing engineering knowledge in the area of process automation and control. AutomationML is becoming widely accepted in industry. It is standardized as IEC 62714. For more information about this format we refer the interested reader to the website of the AutomationML Office1. For more information about this format we refer the interested reader to the website of the AutomationML Office1.

AML may be considered as data integration format for the following standardized data representations: CAEX for plant topology information, COLLADA for geometry and kinematic information, and PLCopen XML for logic information. The topology description is captured in the CAEX-related part of AML. The entire plant topology model is represented as an “instance hierarchy” in AML. Devices of the plant are represented as so-called “internal elements” of the aforementioned instance hierarchy or nested in other internal elements. The type of a device is represented as “system unit class”, since an internal element is in this case an instance of a system unit class. The interconnections between devices are represented as “internal links”. For expressing the meaning of captured information, AML defines “role class libraries”, which should be shared among various projects. Interfaces of artifacts are modeled with “interface class libraries”. The remaining parts of AML, i.e., PLCopen and COLLADA are not used in this paper and therefore not further introduced.

To utilize the benefits offered by modern model-driven frameworks [13] for AML, we have developed a model-driven engineering workbench for AML [7] based on the Eclipse Modeling Framework (EMF) [14]. In particular, we have formalized the CAEX language in a metamodel using EMF’s metamodeling language Ecore, which enables the utilization of EMF’s rich ecosystem of model-driven tools for AML. The developed AML workbench is publicly available in our source code repository [7]. Of course, query languages available for EMF such as the Object Constraint Language (OCL) may be directly employed for querying AML models. Although such query languages offer powerful query concepts and mechanisms, as we will discuss later, for domain experts it is challenging to use such general query languages to formulate queries. In this paper, we use our previous work as a basis for implementing the presented AML query approach AQL. In particular, we reuse the AML metamodel to derive the query definition language as well as the query result language.

B. Current Query Support for AutomationML

As AML comes with XML-based standards, one possible way to define queries is to use XML-based query languages such as XQuery, XPath, or XSLT. However, these languages heavily operate on the abstract tree structure behind the AML models and require extensive knowledge in XML processing models such as path expressions, axis navigation, etc.

Another approach is to use the APIs which are currently available in AML engines2 to query the models by using general purpose programming languages such as C# and Java. However, this approach also requires knowledge on the structure of AML models behind the APIs as well as knowledge about the APIs and the used programming languages. Moreover, the queries have to be mostly formulated in an imperative way instead of using a more declarative query language.

In the AML Analyzer approach [6], AML models are represented as RDF knowledge graphs. This allows to apply query languages for RDF knowledge graphs such as SPARQL. Again, in this query approach knowledge is required about the encoding of AML models in RDF knowledge graphs before the queries can be formulated given also some knowledge on using the SPARQL query language. Similar possibilities and limitations arise when hosting AML models in databases, e.g., SQL or NoSQL based ones. For instance, a first proposal for hosting AML models in NoSQL databases is discussed in [15].

C. Query By-Example

Our proposed AML query approach follows the main principles of the query by-example (QBE) approach introduced in [10], [11]. Initially, the aim of QBE was to have a language for querying and manipulating relational data. This is achieved by so-called skeleton tables, which consist of example rows filled out with constants, constraints, and variables, combined with commands. Commands describe what to do with the selected tuples that match the defined queries, such as deletion or selection of tuples. In order to operate on relational data stored in DBMS, technology-specific queries (e.g., SQL scripts) are derived from the skeleton tables and can be executed on relational models.

The main motivation behind QBE is to provide a more natural interface for end-users to formulate queries. The main assumption is that stating as an example the result of a query is easier to define compared to writing the query in a more computation oriented language.

If we assume to host AML models in relational databases for which QBE is available, we would still require knowledge about how AML models are persisted in relational databases which may come with a huge impedance mismatch. Therefore, we aim in this paper for a dedicated by-example query language which does not require knowledge on how AML models are encoded in specific technologies but directly operates on the AML concepts and structures. The resulting language and dedicated tool support is presented in the following Section 3.

Other by-example approaches related to our proposed query approach are often summarized under the term programming by-example (PBE) [16]. The objective of these

---

1https://www.automationml.org
2https://www.automationml.org/o.red.c/tools.html
approaches is to facilitate the end user to be able to perform tasks which normally need more knowledge, e.g., knowledge about programming languages. The way PBE tries to achieve this objective is to record the users actions (e.g., using traces) maybe in more than one iteration, and generate a program from the traces to automatically perform the above manually performed task by the computer. We consider this line as interesting future work not tackled in this paper. For instance, PBE may be used to derive the query specifications presented in this paper by recording selection actions of the users in the model editor.

III. APPROACH

In this section, we introduce our approach for extending the AML metamodel with query capabilities in order to model graph pattern queries as AML model fragments. Therefore, we shortly describe an excerpt of the core part of the AML metamodel. Then we introduce our proposal for AQL consisting of two main parts: (i) the AML Query Definition Language (AQDL) and (ii) the AML Query Result Language (AQL). Finally, we describe the prototypical implementation of AQL.

A. AutomationML Metamodel

For explaining our approach, we use a small fragment of the AML metamodel which covers some core concepts such as the instance hierarchy, internal elements, and system unit classes. Thus, with this excerpt we can model the structure of a system and type the system elements with system unit classes. Fig. 1 depicts this excerpt UML class diagram notation. Fig. 2 illustrates a small example which is using these concepts on the model level. The figure models just a small part of the Pick and Place Unit (PPU) [17], [18] hosted at the Institute of Automation and Information Systems at the Technical University of Munich.

B. A Graph Pattern Query Metamodel

In order to perform graph pattern queries, some dedicated concepts are needed to define query models which can be evaluated on base models. As we are aiming for a by-example query language, we do not define a query language on its own, but some query concepts which can be combined with AML. In particular, graph pattern queries can be seen as model fragments which mostly comply to the base modeling language. In other words, we embed some query concepts to the base modeling language AML. It may be seen as the inversion of general query languages which constitute new languages and embed the base modeling languages as part of their type systems.

Our proposed graph pattern query modeling language is shown in Figure 3. Its central concept is the QueryObject which provides attributes needed for defining graph queries such as matching for non-existence or existence (cf. attribute negated) or specifying which elements should be returned by the query (cf. attribute returned). Please note that this class is abstract, and thus, only provides additional attributes for the modeling concepts of the base modeling languages. Therefore, it is intended to let all base language concepts inherit directly or indirectly from this class. By this mechanism, the modeling elements become query elements as we will now show for AML.

C. Augmenting AutomationML with Query Capabilities

Having the AML metamodel and the graph pattern query metamodel at hand, we are now discussing how the resulting AQL in particular, the AML Query Definition Language (AQDL) part, looks like. The AML Query Result Language (AQL) is discussed in the following subsection.

Fig. 4 shows the outcome of merging the AML metamodel with the query metamodel. As we can see in this figure, it is now possible to define queries with AML concepts such as internal elements and system unit classes as they are now query objects. We can even query for CAEXObjects which are the unification of all other AML concepts shown in this metamodel excerpt as we will see later in Section 4.

Please note that in addition to the merging of the two metamodels, we may make abstract classes as concrete ones. By this, we can formulate more general queries. Furthermore, we may have to relax some multiplicities as we are not modeling complete models, but only model fragments for defining the queries. This means, if we have lower bound multiplicities greater than zero for properties, such as for the
name attribute of the CAEXObject class, we have to reset them to zero. Otherwise, we would have to introduce more information as needed for defining a query.

D. Explicitly Representing Query Results

Having AQDL, a language to explicitly model queries, at hand, an interpreter is able to execute the queries for particular AML models. For representing the results of the query execution, we provide an additional sublanguage of AQL, named AML Query Result Language (AQRL) which allows to represent the query results as proxy elements to the input model elements.

Fig. 5 shows the proposed language AQRL. By means of this language, we can document for which input models the query models have been executed. Furthermore, the main content of the query result models are the matches which are computed for each query. A match defines for each QueryObject which should be reported the actual binding, i.e., the matched objects in the AML model.

E. Prototypical Implementation in Eclipse

Based on our previous work for providing tool support for AML in Eclipse [7], we implemented a prototype of AQL. In particular, we specified AQDL and AQRL as Ecore-based metamodels. Using the standard EMF capabilities, we generated tree-based modeling editors for both languages. For executing AQDL queries on AML models, we implemented a prototypical interpreter in Java. The interpreter reads the AML models as well as the AQDL models and produces AQRL models as output. We provide an open source implementation of our prototype with further description and examples on our project website3.

In Fig. 6, we provide a screenshot of our prototype showing from left to right: an AML model, an AQDL example query, and the result of the query as an AQRL model. This example is specifically showing a simple query, namely retrieve all internal elements. This query is defined in the AQDL model by stating a QueryObject that should be matched for internal elements. Thus, the QueryObject is represented by an internal element without any further constraints which acts as a template for the query. The computed AQRL model is providing three matches for the given query and the given model. In the right properties window, the result object for the first match is shown, i.e., the Stack internal element which is the first internal element in our example instance hierarchy (cf. left modeling editor).

In the following section, we describe several examples of queries which can be defined, executed, and the results represented in our prototypical implementation.

IV. AUTOMATIONQL BY-EXAMPLE

After introducing the language definition of AQL, we now demonstrate the language for the example model shown in Fig. 2. In particular, we instantiate each language feature by defining a specific query (Q) requiring this feature. Fig. 7 illustrates the queries and their results for the example model in pseudo notation. We use the identifiers of the base model elements (1-11) to mark if the element is contained in a query result or not.

Q1: This query is just containing a single query object, symbolizing that we want to find all CAEXObjects. Since we are interested in indirect instances as well, we mark this query object as deep to match for indirect instances. The result of the query is the set of all elements of the example model, since they are all indirect instances of the class CAEXObject.

Q2: This query searches for internal elements which have the weight attribute set with a value greater than 50. We mark both query objects to be reported and add constraints for the name attribute of the attribute as well as for the value attribute. The result of executing the query is a tuple which represents the internal element and its contained attribute, since only one internal element has a weight attribute defined, but it also fulfills the value constraint.

Q3: This query searches for all internal elements which are direct child nodes of the PPU instance hierarchy, i.e., more precisely, all instance hierarchies with the name “PPU”. The result of executing this query for our example is a triple which represents these internal elements: Stack1, Crane1, and Ramp1.

Q4: This query selects all internal elements of the PPU instance hierarchy which have at least another internal element as a child node. The result includes two entries: the internal element Stack1 with its internal element Conveyor1 and the internal element Stack1 with its internal element Store1. Please note that in this case, the entries are tuples as we return for a match more than one element.

Q5: This query searches for all internal elements of the PPU instance hierarchy which have another internal element
as child and if so all further internal elements down the hierarchy are included in the result. For this, we use the transitive operator which computes the transitive closure of the reflexive containment reference of internal elements. The result of this query contains two tuples. The first tuple contains the internal element Stack1 with its internal element Conveyor and the second tuple contains the internal element Stack1 with the internal elements Store1 and it’s internal element Sensor1.

Q6: This query searches for all internal elements which have no further internal elements as a child node, i.e., we want to retrieve all leaf nodes. The result set of this query shows the internal elements Conveyor1, Sensor1, Crane2, and Ramp3.

Q7: This query selects all internal elements that refer to a system unit class (SUC) named “Stack”. The result is the internal element Stack1 of the system model which references to the SUC Stack of the prototypes library.

Q8: Finally, this query searches for all internal elements which have more than two internal element as direct child nodes. For this, we use the multi object feature of our query language which directly matches for a set of elements at once. The result of this query is the empty set for our example.

V. CRITICAL DISCUSSION

The shown examples in the previous section illustrate that there is a class of queries which can be defined by augmented AML model fragments, i.e., by visualizing the inherited attributes from the QueryObject class for AML model elements.

Following this approach, we can mostly reuse the modeling style of AML for defining queries. This means, we instantiate elements from the AML language to represent what should be reported by a query. However, we also have to define conditions by adding expressions instead of concrete values to the properties of AML. Although this is still within the structures of AML, the expressions have to be defined with an expression language instead of “just” providing values. However, our hypothesis is that adding an expression language for this purpose, results in shorter code fragments as directly formulating the full query with expression languages.

Our current query definition and query result editors are not providing an optimized syntax for modeling the queries as we have done with our pseudo notation, e.g., cf. Fig. 6. Especially for representing the query results to end users, directly highlighting model elements in the AML editors may be preferred and interactive techniques should be provided to explore the matches which have been computed. Our current structured representation of the query results may be interpreted by dedicated user interface components as we provide a dedicated model structure. Another alternative is to provide model transformations to generate summaries of the query results which can be customized by the users.

Finally, we provide a first version of the query interpreter which is directly executing the query models. However, due to performance and scalability requirements, the query models may be translated to existing query engines which provide support for optimizing queries out-of-the-box. While we currently see the interpreter approach more suited to experiment with query language concepts, for industrialization concerns, a generator approach to target powerful query engines is preferable and subject to our future work.

VI. CONCLUSION AND FUTURE WORK

In this paper, we have presented a novel approach for querying AML models by-example. We presented AQL, a graph pattern based query language which is based on AML. AQL supports positive and negative graph patterns,
computing transitive closures to investigate recursive tree structures, and to match for element sets. The query results are explicitly represented in a result model which acts as a proxy to the AML base model elements.

While already several interesting queries can be formulated in AQL, further work is required to fully exploit the potential of a by-example query language for AML. In the future, we plan to explore further application cases of the developed query language as well as how to integrate other query concepts such as ordering and aggregation. In particular, we are interested in the evaluation of existing query (by-example) languages with respect to expressivity. Moreover, for providing a scalable evaluation of the queries, transformations to existing query engines such as EMF IncQuery\(^4\) may be of interest to exploit sophisticated techniques such as query optimization and incremental evaluation. Finally, expanding the query by-example approach to a transformation by-example approach [19] for AML is another research line. We are already able to query model information which may be used in the future by transformation templates to introduce new elements, or to delete and modify existing ones.

\(^4\)https://www.eclipse.org/viatra/query.php

ACKNOWLEDGMENT

This work has been funded by the Austrian Federal Ministry of Science, Research and Economy and the National Foundation for Research, Technology and Development. We thank Prof. Birgit Vogel-Heuser and her team for providing detailed documentation about the Pick and Place Unit (PPU).

REFERENCES