Software and Systems Modeling (2022) 21:1643–1664
https://doi.org/10.1007/s10270-021-00962-2

REGULAR PAPER

Using two case studies to explore the applicability of VIATRA for the model-driven engineering of mechatronic production systems

Gennadiy Koltun[1] · **Mathis Pundel**[1]

Received: 4 July 2020 / Revised: 15 November 2021 / Accepted: 24 November 2021 / Published online: 21 February 2022

Abstract

The engineering of mechatronic production systems is complex and requires various disciplines (e.g., systems, mechanical, electrical and software engineers). Model-driven engineering (MDE) supports systems development and the exchange of information based on models and transformations. However, the integration and adoption of different modeling approaches are becoming challenges when it comes to cross-disciplinary work. VIATRA is a long-living enduring and mature modeling framework that offers rich model transformation features to develop MDE applications. This study investigates the extent to which VIATRA can be applied in the engineering of mechatronic production systems. For this purpose, two model transformation case studies are presented: "SysML – AutomationML" and "SysML4*Mechatronics* – AutomationML." Both case studies are representative of structural modeling and interdisciplinary data exchange during the development of mechatronic production systems. These case studies are derived from other researchers in the community. A VIATRA software prototype implements these case studies as a batch-oriented transformation and serves as one basis for evaluating VIATRA. To report on our observations and findings, we built on an evaluation framework from the MDE community. This framework considers 14 different characteristics (e.g., maturity, size, execution time, modularity, learnability), according to the *Goal-Question-Metric* paradigm. To be able to evaluate our findings, we compared VIATRA to ATL. We applied all cases to a lab-size mechatronic production system. We found that, with VIATRA, the same functions for model transformation applications can be achieved as with ATL, which is popular for model transformations in both the MDE and the mechatronic production systems community. VIATRA combines the relational, imperative, and graph-based paradigms and enables the development and execution of model-to-model (M2M) and model-to-text (M2T) transformations. Furthermore, the VIATRA internal DSL is based on Xtend and Java, making VIATRA attractive and intuitive for users with less experience in modeling than in object-oriented programming. Thus, VIATRA leads to an interesting alternative for the model-driven engineering of mechatronic production systems. It has the potential to reduce the complexity during the development of model transformations. To conclude, this paper evaluates the applicability of VIATRA, its strengths and limitations. It provides lessons learned and insights that can stimulate further research in the MDE for mechatronic production systems.

Keywords Model Transformations · Model-Driven Engineering · VIATRA · Mechatronic Production Systems · Applicability Study

1 Introduction

The development of mechatronic production systems is highly complex and, therefore, requires the collaboration of different stakeholders. To manage this complexity, Model-Based Engineering (MBE) is an application field that combines a variety of discipline-specific and interdisciplinary models [1,12].

Model-Driven Engineering (MDE) represents a subset of MBE [12] in which metamodels and transformation engines are integrated to exchange information based on models [50].

Communicated by Jeff Gray.

✉ Gennadiy Koltun
Gennadiy.Koltun@tum.de

Mathis Pundel
Mathis.Pundel@tum.de

[1] Department of Mechanical Engineering, Technical University of Munich (TUM), Boltzmannstr. 15, 85748 Garching, Munich, Germany

Transformations represent the core of MDE, which is why a great variety of model transformation approaches (languages and tools) has been developed (cf. [15,37]). Kahani et al. [37] refer to an African proverb that long-living and mature approaches will achieve higher acceptance and applicability when they are maintained and supported over a long period of time.

VIATRA [8] is a mature open-source software project that has existed for more than two decades [59]. It supports extensive model transformation features that allow the development of efficient model-driven applications. Although a variety of research in academia and industrial practice has been conducted, to our best knowledge there is no existing study that investigated the applicability of VIATRA in the context of mechatronic production systems. In contrast, we recognized that often ATL is applied. These facts motivated the authors to conduct this study and to address the following research questions: *(1) How can VIATRA be used when being applied to the small, representative model transformation case studies from the engineering of a mechatronic production system? (2) What findings (e.g., compared to ATL) can be derived for practitioners?*

To answer this research question, we investigated the applicability of VIATRA in a lab-size context. This paper provides the following contributions: (i) To be representative for the model-driven engineering of mechatronic production systems, we present two model transformation case studies "SysML – AutomationML" and "SysML4*Mechatronics* – AutomationML" (Sect. 4). These case studies have been widely studied [7,38,43], and we have adapted them to fit the purpose of our study. (ii) A VIATRA software prototype implemented the case studies as batch-oriented transformations, i.e., models are built from scratch (Sect. 5). (iii) We propose an evaluation framework, which was inspired by [39], to assess VIATRA. Within this evaluation framework, 14 different characteristics (e.g., maturity, size, execution time, modularity, learnability) were considered. To provide a reference for our findings, we related VIATRA to other model transformation approaches.

The remainder of the paper is structured as follows: Sect. 2 presents the relevant background. Sect. 3 details the objectives of the study. Section 4 provides an overview of the model transformation case studies, and Sect. 5 presents implementation details regarding the software prototypes. Section 6 provides the evaluation results on the applicability of VIATRA, and Sect. 7 discusses threats to validity. Finally, Sect. 8 concludes this paper and presents an outlook on future work.

2 Background

In this section, we will provide a brief overview of the cross-disciplinary engineering in mechatronic production systems (Sect. 2.1). Based on that, we introduce heterogeneous model-based engineering in the field of mechatronic production systems (Sect. 2.2). We then explore the relevant literature about model transformation approaches, e.g., VIATRA and ATL (Sect. 2.3). For a detailed overview of model transformation approaches and features, please refer to [15,37,63].

2.1 Cross-disciplinary engineering of mechatronic production systems

The engineering of mechatronic production systems requires the strong collaboration of different disciplines, e.g., systems engineering, mechanics, electrics/electronics and control software. According to Berardinelli et al. [7] and Vogel-Heuser et al. [61], mechatronic production systems are highly complex systems that consist of different mechanical, electrical hardware and automation software (see Fig. 1). To develop those mechatronic systems, the successful integration of all technical components and disciplines involved are critical success factors. To foster an efficient and effective engineering process of mechatronic production systems, there is a continuous demand for models, modeling languages, methods, and tools. Their development, integration and adoption constantly bring new challenges (e.g., consistency, interoperability, communication, sustainability, applicability, etc. [1]), which results from interdependencies between models and the associated challenges, e.g., interoperability, inconsistency, reasoning, traceability, or verification (cf. [24,25]). To overcome these challenges, developers (hereafter referred to as the MDE community) and practitioners (hereafter referred to as the production systems community) must understand each other to collaborate fruitfully and develop tightly harmonized solutions of models, modeling languages, and tools. In the authors' view, this is a continuous process. In the following Section, we introduce four application scenarios for model-based engineering and how they help to overcome challenges mentioned in this section.

2.2 Model-based engineering of mechatronic production systems

Model-Based Engineering (MBE) is essential for managing complexity in the development of mechatronic systems. Various stakeholders from different disciplines make use of models in terms of modeling languages and tools [13,53]. In the following, we will briefly present four main application scenarios with respect to MBE.

Model-Based Systems Engineering (MBSE) is an approach that fosters interdisciplinary collaboration during systems development [27]. Different models are intended to address distinct challenges, e.g., requirements specification, struc-

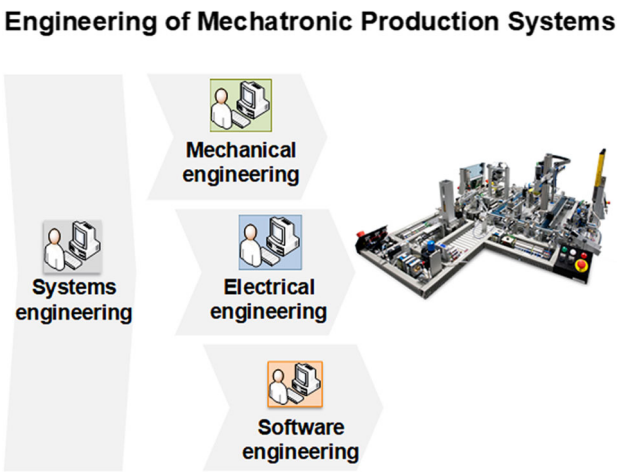

Fig. 1 Simplified interdisciplinary engineering of mechatronic production systems

tural and behavioral description of the system under investigation, simulation, and others [27]. These models must be integrated and linked along the entire lifecycle [13,35]. MBSE is a process where models play an important role. In Model-Driven Engineering (MDE) are models the key artifacts to transform model or to generate software code [12].

Model-Driven Architecture (MDA) is often referred to in the context of MDE and is specified by the object management group (OMG). Within MDA [47], models are instances of metamodels. Metamodels are required to represent the relationship between a class and its instance. A metamodel can be seen as a model, and it allows one to define a modeling language. To provide an architecture in which metamodels are used, the OMG defines the *Meta Object Facility* (MOF) [50]. When applying MOF, it is often referred to as a four-layered metamodel architecture: M0 represents real-world objects and corresponds to the M1 model, which is the model for the reality. The M1 model corresponds to a M2 model, which is a metamodel. A metamodel (M2) corresponds to a so-called meta–metamodel (M3), which represents a platform-independent model. In other words, M3 provides the underlying modeling pillar used for defining M2 metamodels.

Domain-specific modeling languages are used for representing particular theories and concepts developed for a specific application domain [26]. Two well-known and standardized domain-specific modeling languages are UML [49], for software engineering, and SysML [51], for systems engineering. Since these standardized modeling languages lack acceptance and applicability in practice [1], extensions of these modeling standards in terms of profiles (e.g., [38,46,54]) or new domain-specific modeling languages (e.g., [6]) are evolving. Hölldobler et al. [28] summarize these approaches as "language derivation" where abstract and concrete syntax of the base language are reused to overcome the challenge of applicability in practice. To conclude, the practice of designing valuable and applicable domain-specific modeling languages is tremendously important in the industrial practice of MBE.

Information exchange among disciplines and tools is another application scenario that aims to improve the efficiency and effectiveness of interdisciplinary collaboration [7,11,43]. A variety of exchange formats exist in order to exchange information between different disciplines and tools. STEP [34] is one standard that specifies an application protocol for the representation of systems engineering data. AutomationML (AML) [31] is an emerging standard within the mechatronic production systems domain that supports the exchange of engineering data based on the XML language. Since engineering data relate to one another, these exchange formats must provide different modeling capabilities in order to capture semantic relationships.

2.3 Model transformations and metamodels

The application scenarios mentioned before might be part of a MDE environment. In MDE, metamodels and transformation engines are integrated in order to bridge domains based on models [53]. Both *model transformations* and *metamodels* have achieved high attention in the field of software engineering by the *Object Management Group*, which is a well-known consortium providing standards such as UML [49], SysML [51] and MDA [47].

Model transformations are important for performing various operations on a set of models (M1), i.e., models might be read, created, or modified [63]. The basic principle of a model transformation is shown in Fig. 2. A *transformation engine* executes a model transformation. The definition of a model transformation is described within a *transformation specification* [15]. Therein, it is referred to as the metamodels (M2) involved in order to provide a mapping between the source and target of a transformation. The transformation specification must conform to the *model transformation language* in a way similar to models (M1) and metamodels (M2).

In practice, a model transformation language can only be applied within a *technical space*, i.e., a model management framework [9]. In other words, the metamodel behind a model transformation language represents the abstract syntax, whereas the concrete syntax of a model transformation language is specified by the technical space [37]. A large variety of model transformation languages and tools have been developed to provide different capabilities for developing diverse model transformation applications. In the following, we will briefly introduce the model transformation approaches by VIATRA and ATL because they are central elements of this article.

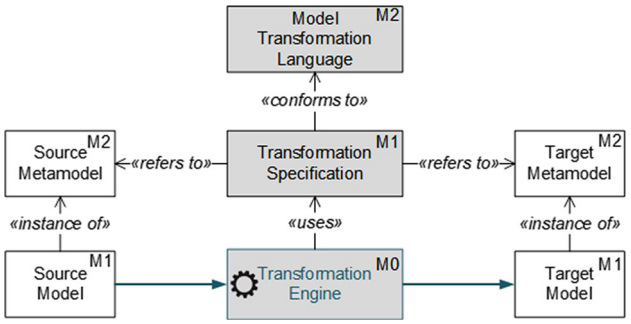

Fig. 2 Basic principle of model transformations, based on [15]

2.3.1 VIsual automated model tRAnsformations (VIATRA)

VIATRA is an open-source Eclipse software project that was developed more than two decades ago and is now in its third generation [8,59]. In addition to the development of batch and incremental model transformations, VIATRA supports model querying, model obfuscating for the confidential treatment of information, and design exploration capabilities for systems engineering [22]. The VIATRA project is tightly integrated into Eclipse IDE and the *Eclipse Modeling Framework* (EMF) [55], in which case it can be combined with other powerful MBE and MDE applications.

VIATRA *batch* transformations satisfy the basic transformation principle shown in Fig. 2. It enables the combination of relational, imperative, and graph-based paradigms [37] in order to specify the mapping between two metamodels of different domains. To perform various operations on a model, VIATRA provides an *Incremental Query Engine* [59] together with the *VIATRA Query Language*, which is a declarative graph-based query language. A transformation specification in VIATRA can be divided into two parts [22]: *rule specifications* and *execution schemes*. Rules refer to model queries describing the operations that need to be performed on a model. The execution scheme defines the transformation behavior, which the *rule engine* executes. To describe a transformation specification, VIATRA provides an internal DSL as a model transformation language based on Java and Xtend [23]. Exemplary VIATRA code listing will be presented in Sect. 5. For details about VIATRA and further model transformation features, please refer to Varró et al. [8].

2.3.2 ATL transformation language (ATL)

ATL is a mature model transformation approach [36] and Eclipse Project [19] is from the field of MDE. The ATL framework provides an environment for the development of MDE applications together with different model operations, such as editing, compiling, executing, and debugging [36]. Given that ATL is completely compliant with OMG's MOF architecture and is widely accepted, it can be considered as the de-facto standard in MDE.

ATL transformations work following the basic transformation principle shown in Fig. 2. To specify the mapping between two metamodels of different domains, ATL allows for combining relational and imperative paradigms [37]. ATL relies on the *Object Constraint Language* (OCL) [48], which is a standardized declarative language provided by the OMG. A transformation specification in ATL is organized using *modules*, which consist of *helpers* and *transformation rules*. Helpers factorize code, like methods in object-oriented programming. Different types of transformation rules allow one to specify the transformation behavior. Exemplary ATL code listing is later presented in Sect. 5. For details about ATL, please refer to Jouault et al. [36].

3 Related work and study objectives

The section is structured as follows: In Sect. 3.1, we first give a brief overview of related work to explain why we are investigating with VIATRA for the model-driven engineering of mechatronic production systems. In Sect. 3.2, we then formulate our two main objectives of this study.

3.1 Related work

With regard to MDE, a multitude of research work has been conducted. In the following, we provide a brief overview of MDE in the industrial context. Straeten et.al. [56] and Whittle et. al. [64] provide two reviews on the industrial applicability of MDE. Berardinelli et al. [7] provide a model transformation between AutomationML (which is an emerging information exchange standard within the mechatronic production systems) and SysML (which is de-facto for cross-disciplinary engineering). Their transformation relies on ATL. Balasubramanian et al. [4] provide graph-based model transformations using the GReAT transformation tool. Lano et al. [42] present an approach for model transformations with UML-RSDS, which combines model transformations with general software systems to design UML models and generate code automatically. Westfechtel [62,63] provides a bidirectional model transformation approach (rather than two unidirectional ones) based on QVT-R. The interesting approach has been validated on formal Petri Nets, rather than on models for describing systems. Shah et al. [54] provide a framework to transform and exchange information between a common SysML model and tool-specific models from EPLAN and Modelica. Note, models such as Modelica and Simulink models are often referred to in a context other than model-based systems engineering. The authors identify tool-interoperability as one major problem that requires the definition of model transformation and intermediate models.

Varró [58] provides an overview of industrial applications of VIATRA3, but none of these are mechatronic production systems. His use cases are exclusively software applications, while mechatronic production systems are characterized by the combination of mechanical, electrical and software components.

To conclude, applying MDE is often less a challenge of technological possibilities than making MDE understandable and easily accessible for a particular practitioner community. There is currently no study that has evaluated VIATRA specifically for the engineering of mechatronic production systems to the best of the authors' knowledge. The study's objectives are explained in the following.

3.2 Study objectives

The research questions (cf. Sect. 1) asked by this study whether and how VIATRA, given its extensive features, can be useful for the model-based engineering of mechatronic production systems. To answer these questions, we have derived two objectives.

Objective 1: Case Study Implementation
With this objective, we aimed to implement small, representative model transformation case studies for model-driven engineering of mechatronic production systems to derive conclusions about the applicability of VIATRA. This objective implies identifying model transformation case studies, corresponding metamodels, and mapping specifications. This conceptual framework serves as a basis for implementing a VIATRA software prototype, while the software prototype allows for verifying the conceptual framework.

Objective 2: Evaluation Framework
With this objective, we aimed to assess VIATRA in a structured manner, i.e., based on an established framework. This includes both the theoretical and the practical evaluation of VIATRA. Furthermore, we aimed to assess VIATRA in relation to other model transformation approaches in order to provide a reference for our findings.

4 Study objects: model transformation case studies

Surveying the applicability of VIATRA for engineers requires appropriate and representative case studies. To put our results and findings into perspective, we have decided to build on other research results. Another benefit of this decision is that these research results rely on instructive experiences of academic and industrial relevance. We gathered important information on the metamodels, the mapping specification, and software prototype insights. We built on a variety of research work and adapted it for this study, i.e., assessing the applicability of VIATRA for model transformation in mechatronic production systems.

The remainder of this section is as follows: In Sect. 4.1, we will present the metamodels of SysML for structural modeling, SysML4*Mechatronics*, and AutomationML (AML). This section provides the basics of the underlying metamodels to understand the transformations built on top of them. In Sect. 4.2, we will present the mapping specifications that form the two model transformation case studies of this study. The first case study, "SysML – AML" is based on the ATL prototype from Berardinelli et al. [7,29]. We adopted this case study for VIATRA. The second case study "SysML4*Mechatronics* – AML" is an extension of the first one and developed solely by the authors of this paper. Thus, we apply VIATRA and ATL to the first case study and only VIATRA to the second case study. As the second use case was developed by the authors, we did not consider it suitable for comparison. Therefore, we only applied and compared VIATRA and ATL for the first case study. Both model transformation case studies form the objects of this study and serve as a basis for the implementation. This section contains enough information to make the paper self-contained and, for detailed information, we refer to the cited literature.

4.1 Metamodels

4.1.1 SysML structural modeling metamodel

SysML is a standardized and graphical general-purpose language used for systems engineering [51]. It reuses parts of UML [49] and provides a powerful extension for modeling various systems aspects such as requirements, structure, and behavior.

In this study, we only considered SysML with respect to modeling the structure of a system. Figure 3 depicts an excerpt of the SysML metamodel. To describe the structure of a system or its components, SysML defines *Blocks*. Each block may contain *properties* and *operations* to describe the features of a component. Since components are interconnected logically or physically, SysML allows for modeling various kinds of semantic relationships (e.g., *associations*, *generalizations* and *dependencies*). At a detailed level of systems modeling, model instances also interact with each other. In this case, SysML allows the modeling of these interactions in terms of *InterfaceBlocks* and *Ports* or *Connections*.

4.1.2 SysML4*mechatronics* metamodel

Although SysML allows one to model the structure of a system, its application becomes inconsistent, and therefore difficult, in practice. Kernschmidt et al. [38] developed SysML4*Mechatronics* for the structural modeling of complex mechatronic systems. As shown in Fig. 4, the

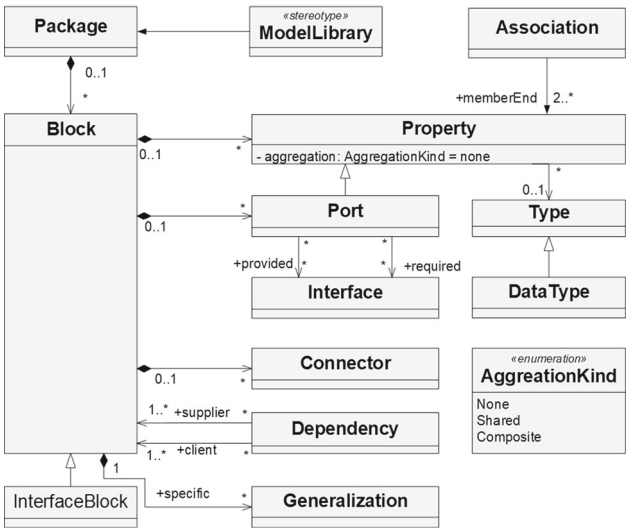

Fig. 3 Excerpt from the SysML metamodel [51] for structural modeling according to [7]

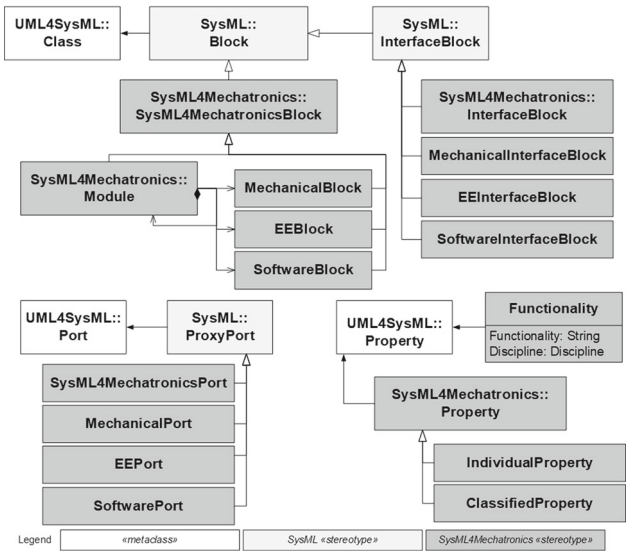

Fig. 4 Excerpt of the SysML4*Mechatronics* profile for the structural modeling of mechatronic productions systems according to [38]

SysML4*Mechatronics* profile extends the core concepts of the SysML metamodel for structural modeling, i.e., it refines blocks, interfaceblocks, ports, and properties. This profile aims to integrate engineering information from a variety of disciplines like mechanics, electrics, and software in order to improve interdisciplinary development processes of mechatronic production systems. For instance, model-based analysis techniques might be intended for approving whether a discipline-specific component is compatible with another one [38].

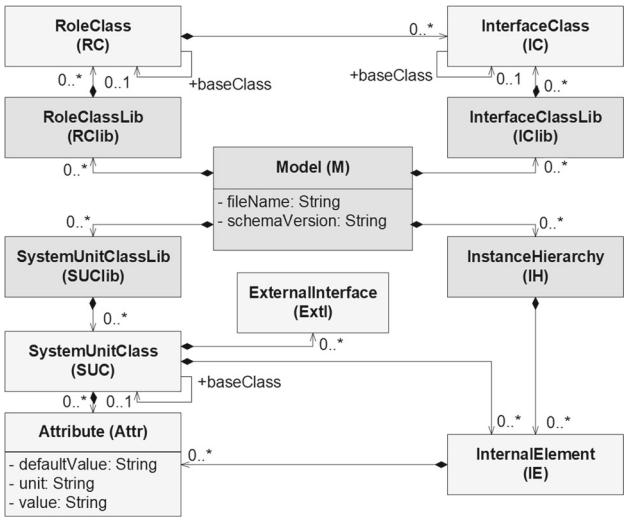

Fig. 5 Excerpt from the AutomationML metamodel [31] for hierarchical plant modeling according to [7]

4.1.3 AutomationML metamodel

AutomationML [31] is an emerging and standardized data exchange format developed for the automation engineering domain. It interconnects information among different disciplines and between heterogeneous engineering tools, e.g., mechanical engineering tools, electrical design tools, or PLC programming tools [7]. AML uses other standards, such as the CAEX standard [30], for describing hierarchical plant structures. It uses COLLADA [5] for describing kinematics, and PLCopenXML [32] as a standardized format for exchanging control software information.

The hierarchical modeling concept of AML is briefly explained in the following and, with respect to AML details, we refer to [7,17,31]. Figure 5 shows an excerpt of the AML metamodel which incorporates four core modeling concepts. (i) The *InstanceHierarchy* represents the engineering project and its components (*InternalElements*); (ii) the *SystemUnitClassLib* defines reusable standard components (*SystemUnitClass*); (iii) the *InterfaceClassLib* comprises discipline-specific and interdisciplinary interfaces (*InterfaceClass*); and (iv) *RoleClassLib* allows one to define semantic relationships between objects. Different kinds of relationships and associations between these elements enable the organization of the data for an engineering project.

4.2 Model transformation case studies

4.2.1 Case study 1: SysML – AML

The first case study, "SysML — AML," is the main focus of this study. We obtained this case study from Berardinelli et al. [7]. One ATL software prototype for this case study already

exists [7,29]. This is beneficial with regard to objective 2 (Sect. 3) since this allows us to provide a reference point for our VIATRA study.

According to Berardinelli et al. [7], SysML represents a class-based modeling language using dedicated diagrams. In contrast, AML represents a tree-based exchange format specifying real-word engineering components of mechatronic production systems. Given that both modeling approaches are based on the object-oriented paradigm, a mapping between these approaches can be derived. Berardinelli et al. [7] provide an *AML4SysML* profile that allows mapping UML/SysML concepts to the AML ones. Table 1 lists the mapping specification and indicates modifications compared to the mapping from [7]. The modifications (lines 1,3,4) were made to avoid ambiguities for both transformation cases, i.e., SysML2AML and AML2SysML. We introduced a different terminology and refined the stereotypes and concepts. Further elements (lines 16,17) are newly introduced to precise the mapping specification. These modifications allowed us to support bidirectional transformations without losing information or manual intervention.

4.2.2 Case study 2: SysML4*Mechatronics* – AML

The second model transformation case study extends the first one by considering the SysML4*Mechatronics* [38] profile. The feasibility of the model transformation "SysML4*Mechatronics* – AML" case study within an industrial toolchain has already been validated by the research of Li et al. [43]. This mapping is restricted to a unidirectional transformation from SysML4*Mechatronics* to the AML exchange format (i.e., the "AML2SysML4*Mechatronics*" direction was not investigated). Furthermore, the mapping proposed in [43] is not based on a profile method as demonstrated with the AML4SysML profile in [7] and the previous section. To overcome these two shortcomings, we extended the "SysML – AML" mapping of the previous section and, at the same time, made it compliant with the mapping presented in [43]. Table 2 lists the "SysML4*Mechatronics* – AML" mapping. As shown in Table 2, the AML modeling concepts have been extended by an additional attribute regarding the mechatronic disciplines (i.e., mechanics, electrics, and software). This allows one to capture the discipline-specific information in the model. All other mappings remain the same as that specified for the "SysML – AML" case study (cf. with Table 1).

5 Study implementation: transformation framework

In this section, we will present details regarding both the implementation and the software prototypes (Sect. 5.1), as well as the test samples used (Sects. 5.2 & 5.3).

5.1 Implementation details

We aimed to survey the applicability of VIATRA for the engineering of mechatronic production systems. For this reason, we focused on *batch* and *bidirectional* model transformations. Since neither VIATRA nor ATL supports bidirectional transformations (cf. [37]), the bidirectional characteristic can be achieved by the composition of two unidirectional transformations: The "SysML – AML" case study considers the mapping specification listed in Table 1 and consists of a "SysML2AML" and an "AML2SysML" transformation. To obtain a bidirectional transformation, the mapping specifications in Sect. 4.2 are essential. For instance, although SysML and SysML4*Mechatronics* are similar concepts, it was a challenge to define the mapping specification as a way to avoid ambiguities. It was a challenge to define the mapping specification to avoid ambiguities. It was essential to define the mapping specifications unambiguously and define the specification as detailed as possible to store all information within a model during a transformation. Only in this way can a loss of information be avoided and, for example, a transformation back to the original model in the source language is realized. However, bidirectional transformation success relies on the model sample. Our study draws on the xPPU demonstrator and model sample based on it, which will be explained later in this section.

To implement the mapping specifications posed in the previous section, a VIATRA software prototype was implemented by using the Eclipse Oxygen IDE. The VIATRA software prototype makes use of the following plugins: The *Eclipse Modeling Framework* (EMF) [20,55] provides a full implementation of the OMG's Meta Object Facility (MOF) [50] by means of the *ecore* metamodel. In addition, *Papyrus* [21] provides a full implementation of the UML [49] and SysML [51] modeling standards. The *Eclipse VIATRA Framework* [8,22] provides the full implementation of VIATRA3 [8], which requires Xtend [23] for its internal DSL.

Table 1 Mapping specification for the model transformation "SysML – ML" case study (the specification is adopted from [7], and modified rows are marked with an asterisk)

	AML Concept	AML4SysML Stereotype	SysML/UML Concept
General			
1*	CAEX File	Model	Model
2	Attribute	Attr	Property
3*	Attribute.Unit	Attr	Attr.Unit
4*	Attribute.Value		Property.Default
Libraries			
5	InterfaceClassLib	IClib	Package "ModelLibrary"
6	RoleClassLib	RClib	Package "ModelLibary"
7	SystemUnitClassLib	SUClib	Package "ModelLibrary"
AML Classes			
8	InterfaceClass	IC, Block	InterfaceBlock/Class
9	RoleClass	RC, Block	Block/Class
10	SystemUnitClass	SUC, Block	Block/Class
AML Objects			
11	InstanceHierarchy	IH, Block	Block/Class
12	InternalElement	IE, Block	Block/Class
13	ExternalInterface	ExtI	Port
Inheritance Relationship			
14	BaseClass	BaseClass	Generalization
Object-Object Relationship			
15	InternalLink	IL	Connector
16*	IL.getPartnerSideA		ConnectorEnd.partwithPort
17*	IL.getPartnerSideB		ConnectorEnd.partwithPort
Object-Class Relationship			
18	RoleRequirement	RR	Dependency
19	SupportedRoleClass	SRC	Dependency
20	BaseSystemUnit	Prototype	Dependency

To provide a reference for the VIATRA software prototype, we additionally drew on the ATL software prototype [29] from the research of Berardinelli et al. [7]. We deployed the ATL prototype within the Eclipse Mars IDE and the following plugins: *Eclipse Modeling Framework* (EMF) [20,55], *Papyrus* for UML/SysML [21] and the ATL tool kit [19], which provide a full implementation of the ATL [36]. Note that the ATL prototype was not used for the second case study ("AML – SysML4*Mechatronics*"), as this was not within the scope of this study.

5.2 Mechatronic production system model sample

The model test sample that we used for both prototypes was retrieved from [7,29].

It represents the *extended Pick and Place Unit* (xPPU), which is a lab-size mechatronic production system demonstrator set up in our laboratory and used in both academic research and teaching [60]. A lab-size demonstrator is a small but representative system [or product] used as a running example to verify and validate the research approach. The xPPU demonstrator consists of various mechanical, electrical, and software components (e.g., conveyor, motor, sensor, and programmable logic controller) that form an overall production system for handling and manipulating different work pieces. Thus, this model sample was suitable for both model transformation case studies. Figure 6 shows an excerpt from the xPPU demonstrator as an AML model and as a SysML4*Mechatronics* model.

5.3 Further model samples of industrial applications

To investigate the generalizability of our approach, we introduce further model samples based on the AutomationML metamodel (cf. 4.1.3). These model samples were retrieved from the AutomationML Association [3] which is supported by many industrial partners and academic institutions from the Mechatronic Production Systems and Automotive Industry. The AutomationML Association [3,16] "provides a comprehensive in-depth look into the practical application

Table 2 Mapping specification for the model transformation "SysML4*Mechatronics* – AML" case study (the specification is based on Table 1, and only adopted rows are listed)

	AML Concept	AML4SysML Stereotype	SysML/UML Concept	SysML4*Mechatronics* Concept
Libraries				
5	InterfaceClassLib	SUClib	Package	SysML4MechatronicsPkg
7	SystemUnitClassLib	IClib	Package	SysML4MechatronicsPkg
AML Classes				
8	InterfaceClass			
	- Mechanical	IC	InterfaceBlock	MechanicalInterfaceBlock
	- EE	IC	InterfaceBlock	EEInterfaceBlock
	- Software	IC	InterfaceBlock	SoftwareInterfaceBlock
10	SystemUnitClass	SUC	Block/Class	Module
	- Mechanical	SUC	Block/Class	MechanicalBlock
	- EE	SUC	Block/Class	EEBlock
	- Software	SUC	Block/Class	SoftwareBlock
AML Objects				
11	InstanceHierarchy	IH	Package	SysML4MechatronicsPkg
12	InternalElement	IE	Block/Class	Module
	- Mechanical	IE	Block/Class	MechanicalBlock
	- EE	IE	Block/Class	EEBlock
	- Software	IE	Block/Class	SoftwareBlock
13	ExternalInterface			
	- Mechanical	ExtI	Port	MechanicalPort
	- EE	ExtI	Port	EEPort
	- Software	ExtI	Port	SoftwarePort

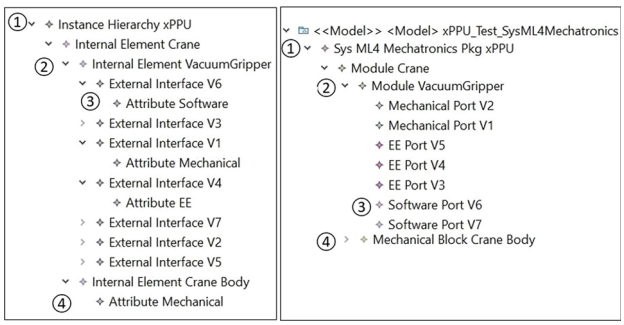

Fig. 6 Model sample of the xPPU demonstrator [60] as an AML model (left) and as a SysML4*Mechatronics* model (right)

of AutomationML Edition 2 from an industrial perspective. It is a cookbook for advanced users and describes re-usable pattern solutions for a variety of industrial applications and how to implement it in software."

The AutomationML Association [3] provides 16 domain model samples for different use case scenarios, which we briefly introduce in the following: (1) a model sample with information for the exchange between process and automation engineering, (2) a model sample for the exchange between different computer-aided engineering (CAE) systems, (3) a model based on the Module Type Package specification, (4) a model with system control diagrams (SCDs) for the Gas Industry, (5) a model with project configurations between ECAD and PLC systems for the Factory Automation, (6) a model with mechanical drive configurations, (7) a model with Material Handling information, (8) a general AutomationML model for component-based production system engineering, (9) a model describing electrical interfaces, (10) an AutomationML Component Checker, (11) a communication network model, (12) a model describing OPC-UA servers, (13) a model describing the Assets Administration Shell specification, (14) a semantic integration model, (15) an extended role class library model, and (16) a model containing enterprise control system integration information. For further details on these models, we refer to the AutomationML Association [3].

In this study, we apply VIATRA and the model transformation case studies (cf. Sect. 4) to these 16 model samples in order to assess the reliability and generalizability (described later in Sect. 6.4).

6 Study evaluation and results

In this section, we will evaluate VIATRA and report our observations. There are many ways of identifying and defining metrics used for evaluating model transformations (cf. [14,39,40,45]). Rahimi [39] provides a comprehensive evaluation framework that is specially designed for assessing model transformation approaches. It integrates the *Goal-Question-Metric* (GQM) paradigm [57] with selected characteristics for the evaluation of software products according to the ISO 9126 standard [33]. Our evaluation builds on the framework of Rahimi [39] because it is a holistic and comprehensive evaluation framework specially developed for model transformations. The structure and adaptations of the framework for our study are briefly explained below.

The *goal* is defined from a specific point of view to address a certain question. The *question* clarifies *what* we aim to evaluate. The corresponding *metric* defines the answers to the question and *how* to measure it. For the purpose of this study, we obtained the goals from Rahimi [39], and the questions were defined according to the scope and objects of this study. Significant changes were made to the metrics since we aimed to investigate the applicability of VIATRA for the development of mechatronic production systems. We omitted ranking points because we were not conducting an in-depth study. Instead, the metrics were used to reflect our observations as objectively as possible and to then derive the findings. Whenever possible, we focused on assessing VIATRA in comparison to other model transformation approaches, so we divided our evaluation into three parts. Table 3 provides an overview of the evaluation conducted and of the remainder of this section.

Table 3 Overview of the evaluation conducted

Evaluation Goals	VIATRA	ATL
Theoretical Evaluation (Section 6.1)		
Abstraction Level	✓	✓
Maturity	✓	✓
Closeness	✓	✓
Prototype Evaluation (Section 6.2)		
Error Handling and Support	✓	✓
Size	✓	✓
Structural Complexity	✓	✓
Execution Time	✓	✓
Completeness and Correctness	✓	
Tool-Interoperability	✓	✓
Modularity	✓	✓
Development Effort	✓	
Survey Evaluation (Section 6.3)		
Understandability	✓	
Learnability	✓	
Attractiveness	✓	
Generalizability Evaluation (Section 6.4)		
Reliability & Generalizability	✓	

6.1 Theoretical evaluation

In this section, we will evaluate VIATRA from a theoretical perspective by comparing VIATRA with other model transformation approaches. We considered publications from the MDE community for this purpose.

Goal: abstraction level

Question: How can VIATRA and other model transformation approaches be categorized?

Metric: According to Kahani et al. [37], model transformation approaches can be categorized as being model-to-model (M2M) or model-to-text (M2T). M2M approaches can be further differentiated: *relational/declarative* approaches focus on *what* needs to be transformed, but not on *how* to realize the transformation; *imperative/operational* approaches focus on *how* to transform without considering structural and semantic relationships; *graph-based* approaches are based on formal algebraic algorithms; and *hybrid* approaches combine the paradigms mentioned earlier. M2T approaches can also be differentiated: *visitor-based* approaches traverse a model according to predefined rules and generated text or code; *template-based* approaches prescribe static text or code, which is filled with information from the input model during a transformation; *hybrid* approaches combine both paradigms.

Observations: VIATRA and ATL are both hybrid M2M transformation approaches: VIATRA allows one to combine relational, imperative, and graph-based paradigms, whereas ATL combines the relational and the imperative paradigm. In addition, VIATRA allows for realizing template-based M2T transformations. Table 4 categorizes further model transformation approaches in order to provide a reference for VIATRA and ATL.

Goal: maturity

Question: Over what period are VIATRA and other model transformation approaches successfully supported and maintained?

Metric: Many model transformation approaches have been developed in recent decades. However, it is only high maturity that leads to high reliability. According to Rahimi [39], maturity is high for an approach that has been supported for more than 8 years and low for one supported by fewer than 4 years.

Table 4 Abstraction level and maturity of VIATRA compared to other model transformation approaches based on [37]

Transformation approach tool	Abstraction level classification	Longevity	Maturity
VIATRA [8,59]	M2M (hybrid) M2T (template)	2000 - 2020	high
ATL [36]	M2M (hybrid)	2005 - 2020	high
Agile UML [41,42] (UML-RSDS)	M2M (relational)	2005 - 2020	high
Kermeta2 [18]	M2M (imperative) M2T (visitor)	2005 - 2012	medium
GReAT [4]	M2M (graph-based)	2004 - 2014	medium
eMoflon [2]	M2M (graph-based) M2M (template)	2006 - 2017	high
Xtend [23]	M2M (imperative) M2T (hybrid)	2013 - 2017	medium- high

Observations: The 2018 research work of Kahani et al. [37] surveyed a large number of model transformation approaches and tools. We assumed that their maturity results were still up to date and built upon them. Table 4 lists the maturity of VIATRA and ATL in comparison to other model transformation approaches. VIATRA has a high level of maturity because it is already in the third generation, and the VIATRA family has been continuously developed since 2000. Nowadays, VIATRA is an open-source project and a part of various industrial and academic tools [58].

Goal: closeness

Question: How close are the transformations to a well-known notation?

Metric: According to Rahimi [39], closeness addresses how familiar one is with the model transformation language. The more familiar the language, the greater the closeness. In mechatronic production systems, the stakeholders are various engineers, e.g., systems, mechanical, electrical, and software engineers.

Observations:

VIATRA uses an internal DSL that is built on Xtend and Java. This makes VIATRA attractive for practitioners with experience in object-oriented programming but less familiar with MDE. Furthermore, VIATRA provides a declarative graph-based query language, *VIATRA Query Language* (VQL), for defining patterns. The use of this query language requires a basic knowledge of graph-theory and expertise with the *Eclipse Modeling Framework* (EMF) [55].

The ATL concept relies on the *Object Constraint Language* (OCL) [48], which is a standardized declarative language provided by the OMG. It is well-known in MDE domains and is flexibly applicable for various purposes: (i) OCL enables MOF-compliant metamodels and model instances to be provided with additional formal constraints and (ii) can be used as a declarative query language for transformations.

We assume that VIATRA can achieve a higher degree of closeness outside the MDE community, whereas ATL is well-known and therefore the de-facto standard in the MDE community.

6.2 Prototype evaluation

In this section, we will evaluate VIATRA as compared to ATL, the software prototypes at hand, and the model sample mentioned in Sect. 5. Of course, the following observations depend on both the transformation and the mapping specifications of the model transformation case studies (Sect. 4). The result is a potential threat to the validity, which we will discuss later in Sect. 7.

Goal: error handling and support

Question: Can the model transformation tools provide useful error messages and appropriate support?

Metric: Error handling and support have an impact on fault tolerance and the robustness of transformations. This metric is high when the model transformation approach offers useful features, e.g., syntax checking and run-time checks. If no mechanisms can be identified, the metric is low.

Observations: During the development phase, VIATRA and ATL provide adequate syntax checking, with the result that faulty implemented transformations cannot be executed. During the runtime phase, VIATRA and ATL provide intensive debuggers that allow the transformations to be checked step-by-step [19,22]. In addition, the VIATRA *Transformation Debugger* consists of a UI component that visualizes transformation-specific information, i.e., model elements related to a rule. Apart from the option for verification, VIATRA and ATL provide validation support. VIATRA

Table 5 Comparison of the non-commented lines of code (size) of VIATRA and ATL

	VIATRA	ATL
Pattern LOC	113	42
Transformation LOC	367	227
Total LOC	480	269

provides a *Validation Framework*. Validation rules and constraints must be implemented in VQL and, during runtime, are approved by a validation engine. In ATL, the language itself provides the validation support since ATL is based on OCL.

To conclude, both VIATRA and ATL rely upon advanced tools, extensive documentation, and broad community support. This might be why both indicated a high level of error handling and support.

Goal: size

Question: What is the size of a transformation?

Metric: The size of the transformation is measured by the non-commented lines of code (LOC) [10,39]. The size can also be measured in kilobytes, but is directly proportional to the LOC and has less informative value or can be misinterpreted. For the size can be argued: The smaller the size, the better that transformation projects can be managed and error susceptibility reduced.

Observations: First of all, we assumed that the transformations were implemented according to the respective rules and guidelines of VIATRA and ATL. That means a linter (source code analyzer) would detect stylistic errors. As a consequence, code lines have not been compressed and human readability preserved. Under these conditions, non-commented LOC is an appropriate metric to assess the size in the scope of this paper. From the size, we can infer the complexity, the development effort and also the maintainability effort. Table 5 shows the size of VIATRA and ATL for the transformation case of "AML2SysML." VIATRA patterns are similar to ATL helpers since they make it possible to factorize code, so they also support reusability. VIATRA patterns are significantly larger than ATL helpers. The transformation itself in VIATRA is also larger than that in ATL. We identified two reasons for this: (i) our VIATRA batch transformation was based on a trace model which represents the mapping between the metamodels involved. This additional trace model might require additional coding effort. (ii) Specifying conditions and implementing imports/extensions in VIATRA were more code-intensive than in ATL.

Goal: structural complexity

Question: How complex is a transformation specification?

Metric: The structural complexity is measured by the number of calls and recursive calls [39]. Low complexity may form the basis for better maintainability and expandability.

Observations: Since the prototypes used were implemented and evolved by different people, comparing the structural complexity of VIATRA and ATL is only possible to a limited extent. The VIATRA transformation consists of 22 patterns, whereas the ATL transformation consists of 20 helpers. Patterns and rules both cause calls during a transformation. In VIATRA, we implemented the pattern rules so as to avoid additional recursive calls. Doing so makes it possible to keep the transformation simple, thus fostering transformation comprehensibility and maintainability. ATL is based on the declarative rule concept, and ATL rules are often referred to as *lazy* and *unique* [19,36]. As a result, recursive calls are inevitable in ATL. In the end, structural complexity is determined by the programming style of the developers, but structural complexity might also have an impact on other objectives like execution time and correctness, which will be discussed later.

Goal: execution time

Question: What is the execution time of the transformations?

Metric: The execution time indicates the duration of the transformations. The shorter the time, the more efficient the transformation. Within industrial projects, long transformations can lead to project delays, which may cause costs and reduce user acceptance.

Observations: We measured the execution time for the "AML2SysML" transformation case with respect to VIATRA and the ATL software prototype. The transformations were conducted on a Windows 10 Pro computer with an Intel © Core™ i7-4600U processor and 8 GB RAM. The execution time was determined for 12 model samples, which we retrieved together with the ATL prototype [7,29]. Although 13 of the model samples were simply for testing purposes, the twelfth model sample represented the xPPU lab-size demonstrator (cf. Section 5). Table 6 indicates the measured execution time.

We observed that transformations in VIATRA were faster for models having a low number of model elements (model samples 1 to 5). In contrast, ATL transformations were faster for larger models (model samples 6 to 12). We analyzed the individual transformations in more detail, i.e., we studied the number of different model types and the number of hierarchies (lines 2,3 in Table 6). We found that imperative rule-calling in VIATRA has a significant impact on the execution time. It slows down the execution time compared to

Table 6 Comparison of the execution time between VIATRA and ATL using the example of the "AML2SysML" transformation case (execution time was measured for different model instances)

Model samples	#1	#2	#3	#4	#5	#6	#7	#8	#9	#10	#11	xPPU
Number of model elements	2	5	10	15	19	40	50	77	67	70	71	305
Number of model types	2	3	3	4	5	6	7	11	11	13	13	16
Number of model hierarchies	2	3	5	6	6	6	6	6	6	6	6	7
VIATRA execution time [ms]	62	95	146	171	165	312	376	506	512	491	569	1232
ATL execution time [ms]	176	216	224	209	225	252	250	331	286	278	334	1481

declarative rule firing in ATL. We also found transformation rules which are more time-consuming than others (cf. model samples 8, 9 and xPPU). This is caused by both the mapping specification and its implementation.

Furthermore, we leveraged batch transformations within our VIATRA software prototype, but we are aware of further VIATRA model transformation features, e.g., reactive event-driven transformations [8]. Although we did not consider these features because we aimed to investigate the applicability of VIATRA for the model-driven engineering of mechatronic production systems, further studies should consider those features to make transformations more efficient.

Goal: completeness and correctness

Question: Do the transformations produce complete and correct results?

Metric: The issue of completeness questions whether all functionalities of the transformation were used to process input models correctly, whereas that of correctness questions whether we were able to generate target models conforming with their metamodels [39]. Of course, high degrees of completeness and correctness are desirable. In the following, we will assess completeness and correctness based on the VIATRA software prototype.

Observations: Concerning completeness, we were able to implement all transformations in a programmatically correct manner according to the mapping specifications posed in Section 4. In the model transformation "SysML–AML" case study, however, we determined model elements (e.g., *external reference* and *version* in the AML) within the xPPU model sample which had not been processed during the transformation. Similarly, this was determined for the model transformation "SysML4*Mechatronics*–AML" case study since the AML*InterfaceClasses* (ICs) and *ExternalInterfaces* (ExtIs) had not been processed. After analyzing the VIATRA transformations, we noticed incomplete mapping specifications.

Concerning the level of correctness, we were able to instantiate all target model elements according to the mapping specifications posed in Sect. 4. However, after evaluating the target model instances, we identified a few model elements that missed a parent-child relationship (composition). The reason for this was found in the way that the transformation was implemented: As mentioned with regard to the evaluation of structural complexity, we preferred a less complex transformation and avoided recursive calls. These calls were necessary in order to create the missing relationships. Manual manipulations were conducted to obtain the desired correctness.

To conclude, complete and correct transformation can indeed be achieved using VIATRA. Nevertheless, optimal transformation results will depend on both the model transformation language and the quality of the mapping specification, as well as the programming approach.

Goal: tool-interoperability

Question: Do the model transformation approaches support standalone applications for transfer to other tool-chains?

Metric: Regarding this goal, we questioned how flexible and adaptable the model transformation approaches to different tools are as established tool-chains exist in industrial practice. In order to successfully transfer model transformation approaches into industrial practice, integration into other tool-chains outside the Eclipse IDE should be possible.

Observations: VIATRA and ATL have been evolving over the long term. As a result, they are part of various industrial and academic tools. Concerning standalone transformation applications, the VIATRA framework provides an API [22] that allows the integration of VIATRA features into any Java application. On this basis, we were able to develop a small standalone Java application and execute transformations outside the Eclipse IDE. We were successful in using the ATLauncher project for executing ATL transformations programmatically [44].

Both VIATRA and ATL transformations can be executed outside the Eclipse tool landscape, but they are limited to Java applications.

Goal: modularity

Question: To what extent is modularity supported?

Metric: According to Rahimi [39], modularity addresses the degree of factorization, i.e., the degree of unique expressions. A high degree of modularity is desirable because this is an essential basis for reusability and extensibility.

Observations: VIATRA by nature provides a modular structure. Graph-based query patterns are thus used during the transformation. During the transformation development for all case studies, we were able to save effort and time by reusing patterns. Applying *preconditions* and *find* constraints to these patterns was particularly valuable for developing robust and reusable transformations.

The nature of ATL also provides a modular structure. Organizing rules into different modules makes it possible to obtain a high level of factorization. ATL *helpers* are similar to the methods used in object-oriented programming. They allow for reusing modules and rules from different points of a transformation. Furthermore, ATL allows one to define declarative rules for processing the source model correctly, which is similar to preconditions in VIATRA.

In conclusion, both VIATRA and ATL provide a wide range of capabilities for achieving a high level of modularity.

Goal: development effort

Question: How long does it take to develop the transformations?

Metric: Development effort is generally measured in a quantitative time phase. The shorter the development time, the more likely the application is in practice.

Observations: Reliable measuring of the development effort becomes challenging because the development effort relies on developer expertise regarding the model transformation language and the complexity of case studies. As a consequence, we reported on our observations during the development process and tried to estimate the development effort. At the beginning of our research, we understood the metamodels and developed the mapping specifications, but we had less practical expertise in VIATRA and ATL. As a consequence, we spent approximately 14 full-time working days learning these model transformation languages and applying them correctly within Eclipse IDE. Developing and implementing the model transformation case study "SysML–AML" took about 16 full-time working days, and the required extensions for the second "SysML4*Mechatronics*–AML" case study required another four working days.

In a nutshell, the development effort for transformations in VIATRA is manageable. We hypothesize that a similar assumption can be made for ATL. We see the reasons for the manageable development effort in the context of wide community support. Furthermore, practitioners with experience in object-oriented programming have significant advantages in applying both VIATRA and ATL.

6.3 Survey evaluation

An experiment with students was carried out to gather feedback about VIATRA. The aim was to provide information on the understandability, learnability, and attractiveness of VIATRA. A sample group of mechatronic engineering and computer science students from bachelor study programs was recruited to participate in the experiment (n = 6, all male). The students were not chosen but rather participated on a volunteering basis. The students had a variety of experience levels in object-oriented programming, but they had experience in neither model transformations nor the model transformation case studies proposed in this paper. The sample size considered was sufficient since our aim was only an initial evaluation of VIATRA.

The experiment was conducted as a tutorial, with a duration of 150 minutes. The experiment consisted of three parts. (i) In the beginning, the purpose of the experiment was explained and the demographic data of the students were collected. (ii) Subsequently, a short introduction to the model transformation approaches with VIATRA was provided in order to provide a general understanding. (iii) In the main part of the tutorial, the students had to complete a VIATRA transformation case study according to step-by-step instructions and execute the transformation correctly. Throughout the entire tutorial, feedback was gathered through a survey and interviews with the supervisor. This provided us with information regarding levels of understandability, learnability, and attractiveness. Please note that, due to the time limit of 150 minutes, the participants did not apply ATL in practice. As a result, we are not able to report on ATL in this section, and this issue should be investigated in further research.

Please note that, due to the time limit of 150 minutes, the participants did not apply ATL in practice. We unfortunately did not have more students to conduct a second survey with ATL. As a result, we are not able to report on ATL in this section, and this issue should be investigated in further research.

Goal: learnability, understandability, attractiveness

Question: How much effort is required to apply VIATRA? How understandable is the written code for prospective engineers? How attractive is VIATRA?

Metric: In accordance with the work of Rahimi [39], levels of learnability, understandability, and attractiveness were measured by means of a survey. During the main part of the experiment, we asked eleven questions regarding learnability, eight questions regarding understandability, and nine questions regarding attractiveness (see Table 7). The assignment of the questions to the categories was not communicated to the students. We used a 7-point Likert scale for each question since our sample set consisted of only six students and we

Table 7 Results of the questionnaire (n = 6; 1 = strongly disagree, 7 = strongly agree)

	Mean	Median	SD
Learnability of the VIATRA Query Language	**4.67**		
It requires less time to learn.	4.83	5	1.47
It encourages me to try out new features.	4.67	4.83	1.21
It does not require remembering many details.	4.83	5.5	1.6
It is easy to remember.	5	5	0.89
It is easy to learn without help or manuals.	2.83	2.5	1.47
The pattern structure facilitates learning.	5.83	6	0.98
Learnability of Transformation Rules	**4.33**		
It requires less time to learn.	4.33	4.5	1.75
It encourages me to try out new features.	4.67	5	2.07
It does not requires remembering many details.	4.5	4	0.83
It is easy to remember.	5.33	5.5	1.21
It is easy to learn without help or manuals.	3.33	3	1.86
Overall Understandability	**5.17**		
It is easy to understand.	5.33	5.5	1.21
It uses understandable terms/concepts.	4.83	3	1.86
It is efficient to apply.	6.17	6	1.51
It is well structured and easy to follow.	5.5	5	1.22
It requires less time to learn.	4	4	1.79
It is easy to remember.	4.67	5.5	1.75
It easy to represent relationships/associations.	5.5	6	1.22
The "Trace Model" supports the understandability.	5.5	6	0.83
Overall Attractiveness	**4.63**		
I would use it myself.	5	5.5	1.79
It is suitable for use in (industrial) practice.	5.67	6	1.37
It motivates me to use model transformations.	5.17	5.5	1.72
It is easy to understand for practitioners.	4.83	5.5	1.94
The "Transform Handler" is easy to use.	4.5	4.5	1.38
A specific GUI would facilitate its use.	5.33	5	1.51
It is suitable for beginners in MDE.	3.5	4	2.17
It is easy for the user to operate in the application.	4.67	5	1.37
It can be used without previous knowledge.	3	3	1.90

aimed to increase the degree of variance in our measurement. The better the survey results, the more likely it is that VIATRA is suitable for MDE environments using mechatronic production systems.

Observations: Although a step-by-step guide was provided, and a supervisor was available for questions, the learning effort was perceived as being slightly high. The main reasons for this were three-fold: (i) limited knowledge about the model transformation case studies and metamodels involved; (ii) no experience with the VIATRA framework; and (iii) little experience with the Eclipse IDE. However, even during the experiment we were able to observe that the participants worked faster and more independently as time progressed. In the interviews, it was reported that the transformation specification using the VIATRA internal DSL was harder to learn than the pattern definition using VIATRA *Query Languages*. The reason for this was that the various rules for specifying transformations require more background knowledge.

VIATRA shows strength in its understandability for both query patterns and transformation specifications. Query patterns explicitly refer to the metamodels. This might be beneficial for domain experts of certain modeling languages who are also the developers of model transformation applications. Further strength in this context was identified when making use of a trace model, which is an EMF-based model that refers to query patterns and thus realizes the mapping specification between two metamodels of different domains. Since this trace model can be inspected graphically either in VIATRA or EMF, the mapping can be more

effectively tracked and understood. Participants asked for additional interactive graphical functionalities that support the development and application of model transformations. Understandability was also perceived when using the VIATRA internal DSL for specifying transformation. The reason for this was the reliance on the internal DSL, which is based on Xtend and Java. As a result, *closeness* (cf. Section 6.1) to an object-oriented programming language has a positive influence on its understandability.

Finally, we asked about the overall attractiveness of VIATRA. It was reported that VIATRA is less suitable for practitioners having little experience in MDE. In contrast, this fact was partly compensated for by the closeness to object-oriented programming. This closeness makes VIATRA attractive for programmers having object-oriented experience to learn and to step into MDE. Another weak point addresses the platform-dependency with respect to the Eclipse IDE, which makes VIATRA less attractive. The perception is that this requires further knowledge. On the other hand, the modularity and the frequent reuse of query patterns make VIATRA attractive. It allows one to develop model transformation efficiently.

In conclusion, VIATRA showed promising strengths and weaknesses that indicate opportunities for improvement. However, we are also aware of factors (e.g., less MDE experience, tool troubles) affecting our findings from this survey. To overcome these factors, our recommendation would be to further validate these findings by conducting in-depth user experience studies.

6.4 Evaluation for reliability and generalizability

Question: How reliable and generalizable is VIATRA and model transformation case studies to other mechatronic production system model samples?

Metric: When validating reliability and generalizability on other mechatronic production system model samples, the following applies: The less change effort and adjustments are required when executing the model transformation, the more reliable and robust are the specified model transformation case studies and the VIATRA solution implemented.

Observations: In Section 6.3, we introduce 16 model samples of industrial applications from the Automation ML Association [3,16]. Most of the model samples were correctly transformed according to the mapping specification of our model transformation case studies. In the following, we discuss and highlight four relevant findings. (i) When importing the model samples into the Eclipse IDE, manual adaptions of the XML file were needed. This step can theoretically be automated, to save costs and time for engineers and model transformation experts. (ii) Few model examples contained *RoleRequirement* elements with an additional attribute. Model elements with those additional attributes (so-called boundary cases) were not considered during the transformation; hence, it implies a loss of information during transformation. The reason for this is not the VIATRA transformation itself, but the non-unique mapping specification combined with our individual implementation of managing boundary cases. When transferring such a solution to industrial practice, such boundary cases must be resolved by defining handling rules in the mapping specification and its correct implementation in VIATRA. (iii) To assess whether a model transformation has generated correct target models, the generated models were inspected manually within the VIATRA-based Eclipse IDE. To simplify and save time for such an inspection process, both VIATRA and Eclipse features might be leveraged, i.e. by developing a customized editor with error messages and visualized traceable model elements. (iv) Due to VIATRA's tool-interoperability, there is a low effort to built-in this implementation into other engineering applications. In conclusion, VIATRA demonstrated their reliable and robust transformation capabilities when applying it to other mechatronic production system model samples. A prerequisite for reliable and robust transformation is an underlying mapping specification that is unambiguous and specifies the handling of boundary cases.

6.5 Synopsis

In this section, we will recapitulate our evaluation and summarize our observations. We assessed VIATRA according to the evaluation framework of Rahimi [39], which makes use of the *Goal-Question-Metric* paradigm [57]. We slightly adapted and applied the evaluation framework for the purposes of our study. Table 8 is a concise summary of our observations regarding the characteristics considered.

Whereas ATL combines the relational and imperative paradigm, VIATRA additionally integrates the graph-based paradigms. Furthermore, VIATRA allows us to develop and execute both model-to-model (M2M) and model-to-text (M2T) transformations. ATL and VIATRA are tightly integrated into the Eclipse IDE and the *Eclipse Modeling Framework*, which allows the development of extensive and industrial transformation applications. At the same time, available interfaces make it possible to couple the application with other external applications. This is especially interesting for the model-based development of production systems. The applications can be developed in the Eclipse IDE but can then be integrated and used in other tool-chains. Which of the two transformation approaches is applied depends not only on the respective capabilities but also on the users' experience. Here we have found that ATL is based on OCL and is closely related to OMG's MDA, increasing the chances to be the favorite of modeling experts. VIATRA, on the other hand, is based on a graph-based query language and an internal DSL based on Xtend and Java, which makes VIATRA

Table 8 Summary of the evaluation results

Evaluation Goals	VIATRA	ATL
Theoretical Evaluation (Sec. 6.1)		
Abstraction Level	M2M (imperative, declarative, graph-based) M2T (template-based)	M2M (imperative, declarative)
Maturity	high (2000-2020)	high (2005-2020)
Closeness	required knowledge in graph-theory, EMF, Xtend/Java, Eclipse IDE	required knowledge in OMG's OCL and MOF, Eclipse IDE
Prototype Evaluation (Sec. 6.2)		
Error Handling and Support	advanced tool-support, extensive documentation and broad community support	advanced tool-support, extensive documentation and broad community support
Size	code-intensive, more difficult to maintain	less code-intensive, clearly represented
Structural Complexity	easy and flexible management of structural complexity	limited management of structural complexity (due to declarative ATL concepts)
Execution Time	faster/slower for small/high number of model elements	faster/slower for high/small number of model elements
Completeness and Correctness	correct programmatical behavior	correct programmatical behavior
Tool-Interoperability	available via VIATRA API	available via additional "ATLauncher project"
Modularity	various concepts available to achieve high modularity, e.g., graph-based patterns, preconditions and constraints	various concepts available to achieve high modularity, e.g., modules, helpers, lazy and unique rules
Development Effort	learn VIATRA $\approx$ 14 full-time working days case study 1 $\approx$ 16 full-time working days case study 2 $\approx$ 4 full-time working days	learn ATL $\approx$ 14 full-time working days case study 1 $\approx$ already available from other research case study 2 $\approx$ not investigated
Survey Evaluation (Sect. 6.3)		
Learnability	VIATRA query languages (VQL) is easier to learn than VIATRA internal DSL for specifying transformations	not investigated
Understandability	graph-based paradigm and closeness to object-oriented programming facilitate understandability	not investigated
Attractiveness	platform-dependency to Eclipse and learning effort are perceived as weak points; modularity and closeness to programming are perceived as strengths	not investigated
Generalizability Eva. (Sect. 6.4)		
Reliability & Generalizability	Applicable to a wide range of model samples; structural transformations are correct and robust; (manual) adaptions for semantic correctness might be needed	not investigated

interesting for users with less experience in modeling but in object-oriented programming. Teaching takes an important role for both transformation approaches to ensure its effective and efficient use for the model-driven engineering of mechatronic production systems. However, since the xPPU represents a lab-size demonstrator and the case studies are specific, generalized conclusions about the scalability are possible to a limited extend. To overcome this threat, we applied VIATRA and the model transformation case studies (cf. Sect. 4) to further 16 model samples from the Automation ML Association [3,16]. We found out that the transformations are structurally correct and robust. However, we foresee adaptions might be needed to ensure semantic correctness. This can be achieved by manual adaptions of the generated (target) models or by adapting the mapping specifications. We discuss this in the threats to internal and external validity (Sect. 7).

In summary, with VIATRA, the same functions for model transformation applications can be achieved as with ATL. Compared to ATL, VIATRA allows the development of both M2M and M2T transformations and this is especially important for the model-driven engineering of mechatronic production systems. VIATRA is characterized by a higher level of closeness due to the graph-based paradigm and its similarity to object-oriented programming. Despite all that VIATRA features, the essential criterion for applicability remains two factors: (i) the previous experience of the user with modeling/programming and (ii) the industrial-specific requirements for a transformation framework. Even though we have identified some interesting findings regarding the applicability of VIATRA, there are also threats to the level of validity that have to be considered when analyzing the evaluation results. These threats will be discussed in the next section.

7 Threats to validity

In the following, we will discuss threats to the validity of this study. We differentiated between the various threat types according to Wohlin et al. [65], i.e., conclusion, internal, construct, and external validity.

7.1 Conclusion validity

Conclusion validity concerns the reliability of the conclusion that is drawn. Our evaluation framework (Section 6) relies on predefined metrics for each goal. However, some metrics might have weaknesses, so the evaluation could lead to either subjective or inaccurate findings. That is why we first reported our observations objectively and then presented our findings. Furthermore, if possible, we assessed VIATRA with respect to other model transformation approaches. Concerning the experiments with prospective engineers (Section 6.3), two retrospective meetings allowed us to prepare the findings. Despite our mitigation strategy, threats to reliability might still remain.

7.2 Internal validity

The degree of internal validity reflects whether the results of a study are trustworthy and meaningful. In other words, it refers to the accuracy of this study. In this study, we identified three factors that might influence the findings thereof. (i) The xPPU as lab-size demonstrator, further 16 model samples and the two model transformation case studies were representative of mechatronic production systems, but there are other demonstrators and transformation case studies between further modeling languages that might lead to deviant findings. To mitigate this issue, our evaluation framework focused on assessing VIATRA. If an impact existed, the affected objectives (e.g., correctness and completeness) are mentioned in Sect. 6.2. (ii) The results of the survey relied on a small sample of prospective engineers, which we classified as representative. Although we conducted short interviews in addition to the survey, there remains the risk that subjective and social factors had an influence on the results. Especially the threat of validity due to the experience level of OO modeling needs to be highlighted. We did not separate between students with limited OO modeling skills and students with high experience in OO modeling which might be leading to subjective and biased results. (iii) VIATRA provides a rich portfolio of model transformation features. Not all of these were included in this study (e.g., event-driven, reactive model transformation), and they may also distort the findings of this study. However, rather than assessing all VIATRA features, this study concentrates on the initial evaluation of VIATRA for the model-driven engineering of mechatronic production systems.

7.3 Construct validity

Construct validity concerns the relationship between the results and the underlying concept. As a result, construct validity considers threats to social factors and the research design [65]. A threat may arise from the motivation of the authors in conducting this study. This social factor is known as *hypothesis guessing* [65], i.e., that the behavior of the authors during this study might be affected (either positively or negatively), depending on their expectations of the results. Another threat arises from the research design of this study. To minimize this threat, our study was based on several research studies. The method we used for conducting the study on model transformations was inspired by [52,63]. The evaluation framework was derived and adapted from [39]. To be representative of the model-driven engineering

of mechatronic production systems, our model transformation case studies were retrieved and adapted from [7,38,43]. By drawing on this research work, we were able to explore the applicability of VIATRA to mechatronic production systems.

7.4 External validity

External validity concerns the ability to generalize the results beyond this scope. The results might be influenced since we restricted ourselves to the two transformation case studies regarding VIATRA batch transformation and a small model sample size. The mitigation strategy in this case was to derive the "SysML–,AML" case study, which is representative of model-driven engineering for mechatronic production systems. To support reliability and generalizability, we extended the case study by considering the SysML4*Mechatronics* profile and by applying VIATRA and the model transformation case studies to further 16 model samples from the Automation ML Association [3,16]. Consequently, this study provides a solid basis for further work. Of course, further case studies are recommended for consideration, in which context the further capabilities of VIATRA can be examined more closely. Another threat arose from the sample of prospective engineers during the survey evaluation (Section 6.3), which might not be representative enough. Our mitigation strategy thereby was to select students having different kinds of programming experience but little to no experience in modeling. This strategy was beneficial for us due to the following two reasons: (i) Many engineers have little experience in modeling (cf. [1]), and (ii) we received valuable feedback on the VIATRA framework together with the Eclipse IDE. Of course, the experience with Eclipse might have biased the results. Our mitigation strategy was here: (1) we held a tutorial with an introduction to Eclipse. (2) Before the survey, we pointed out that tool and transformation language should be evaluated as separately as possible.

8 Conclusion and outlook

This paper presents an applicability study of VIATRA for the interdisciplinary model-driven engineering of mechatronic production systems. Drawing on other research, two representative model transformation case studies, "SysML–AutomationML" and "SysML4*Mechatronics*–Automation ML," are presented and implemented as batch transformations within a VIATRA software prototype. The prototype was evaluated against various criteria and compared with other model transformation approaches, e.g., ATL, which is the de-facto standard for model transformations in MDE. The important findings will be concluded in the following.

Whereas ATL combines the relational and imperative paradigm, VIATRA additionally integrates the graph-based paradigms, which makes VIATRA attractive and intuitive for users with less experience in modeling than in object-oriented programming. Furthermore, VIATRA allowed us to develop and execute both model-to-model (M2M) and model-to-text (M2T) transformations which is extremely important in the engineering of mechatronic production systems. ATL and VIATRA are tightly integrated into the Eclipse IDE and the *Eclipse Modeling Framework* (EMF), which enable the development of extensive and industrial transformation applications. At the same time, the available interfaces make it possible to couple the application with other external applications. This is especially interesting for the model-based development of production systems, since the applications can be developed in the Eclipse IDE, but can then be integrated into and used in other tool-chains. Which of the two transformation approaches is applied depends on both the respective capabilities and user experience. In this case, we found that ATL is based on OCL and is closely related to OMG's MDA, thus increasing the chances of being the favorite of modeling experts. VIATRA, on the other hand, is based on a graph-based query language and an internal DSL based on Xtend and Java, which makes VIATRA interesting for users having less experience in modeling than in object-oriented programming. In both transformation approaches, teaching takes an important role in ensuring effective and efficient use for model-driven engineering of mechatronic production systems.

In a nutshell, with VIATRA, the same functions for model transformation applications can be achieved as with ATL. Additional promising capabilities make VIATRA attractive for the model-driven engineering of mechatronic production systems. Of course, many research efforts still remain, including among them the derivation of complete application cases with model samples representing engineering data from a real-world production system. Corresponding requirements for the transformation framework can be derived and boundary conditions on the company side might be identified. The VIATRA framework is able to make use of different (e.g., reactive and event-driven) model transformation features in order to develop a flexible, adaptable transformation framework that meets industrial requirements. Furthermore, the framework can be integrated into an already existing tool-chain to improve development process efficiency. Finally, a comparative study with this baseline provides information on whether VIATRA in industrial practice is beneficial.

Acknowledgements This work was supported by the German Research Foundation (Deutsche Forschungsgemeinschaft, DFG) collaborative research centre 'Sonderforschungsbereich SFB 768' Managing Cycles in Innovation Processes – Integrated Development of Product-Service-Systems based on Technical Products.

Funding Open Access funding enabled and organized by Projekt DEAL.

References

1. Albers, A., Zingel, C.: Challenges of Model-Based Systems Engineering: A Study towards Unified Term Understanding and the State of Usage of SysML. In: CIRP Design Conf., pp. 83–92. Springer Berlin Heidelberg (2013). https://doi.org/10.1007/978-3-642-30817-8_9
2. Amelunxen, C., Königs, A., Rötschke, T., Schürr, A.: MOFLON: a standard-compliant metamodeling framework with graph transformations. In: Rensink, A., Warmer, J. (eds.) Model driven architecture - foundations and applications, pp. 361–375. Springer, Berlin Heidelberg (2006)
3. AutomationML e.V.: AutomationML (2020). https://www.automationml.org/industrial-application/domain-model/. Accessed on 10 June 2021
4. Balasubramanian, D., Narayanan, A., VanBuskirk, C., Karsai, G.: The graph rewriting and transformation language: GReAT. Third International Workshop on Graph Based Tools **1**, 1–8 (2006). https://doi.org/10.14279/tuj.eceasst.1.89
5. Barnes, M., Finch, E.L.): COLLADA - Digital Asset Schema Release 1.5.0 (2008)
6. Becker, S., Koziolek, H., Reussner, R.: The Palladio component model for model-driven performance prediction. J. Syst. Softw. **82**(1), 3–22 (2009). https://doi.org/10.1016/j.jss.2008.03.066
7. Berardinelli, L., Biffl, S., Lüder, A., Mätzler, E., Mayerhofer, T., Wimmer, M., Wolny, S.: Cross-disciplinary engineering with AutomationML and SysML. At - Automatisierungstechnik **64**(4), 253–269 (2016). https://doi.org/10.1515/auto-2015-0076
8. Bergmann, G., Dávid, I., Hegedüs, Á., Horváth, Á., Ráth, I., Ujhelyi, Z., Varró, D.: Viatra 3: A Reactive Model Transformation Platform. In: Kolovos, D., Wimmer, M. (eds.) Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). Lecture Notes in Computer Science, vol. 9152, pp. 101–110. Springer International Publishing, New York (2015)
9. Bézivin, J., Kurtev, I.: Model-based Technology Integration with the Technical Space Concept. In: Proceeding of the Metainformation Symposium, pp. 1–18. Springer-Verlag (2005). http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.106.1366
10. Bhatt, K., Tarey, V., Patel, P., Mits, K.B., Ujjain, D.: Analysis of source lines of code (sloc) metric. Int. J. Emerg. Technol. Adv. Eng. **2**(5), 150–154 (2012)
11. Biffl, S., Lüder, A., Rinker, F., Waltersdorfer, L.: Efficient engineering data exchange in multi-disciplinary systems engineering. In: P."Giorgini, B. Weber (eds.) Advanced Information Systems Engineering, pp. 17–31. Springer International Publishing (2019). https://doi.org/10.1007/978-3-030-21290-2_2
12. Brambilla, M., Cabot, J., Wimmer, M.: Model-driven software engineering in practice, 2nd edn. Morgan & Claypool Publishers, New York (2017)
13. Broy, M., Feilkas, M., Herrmannsdoerfer, M., Merenda, S., Ratiu, D.: Seamless model-based development: From isolated tools to integrated model engineering environments. Proc. IEEE **98**(4), 526–545 (2010)
14. Cabot, J., Clarisó, R., Guerra, E., de Lara, J.: Verification and validation of declarative model-to-model transformations through invariants. J. Syst. Softw. **83**(2), 283–302 (2010). https://doi.org/10.1016/j.jss.2009.08.012
15. Czarnecki, K., Helsen, S.: Feature-based survey of model transformation approaches. IBM Syst. J. **45**(3), 621–645 (2006). https://doi.org/10.1147/sj.453.0621
16. Drath, R.: AutomationML: the industrial cookbook. De Gruyter Oldenbourg (2021). https://doi.org/10.1515/9783110745979
17. Drath, R., Lüder, A., Peschke, J., Hundt, L.: AutomationML - The glue for seamless Automation engineering. IEEE International Conference on Emerging Technologies and Factory Automation, ETFA pp. 616–623 (2008). https://doi.org/10.1109/ETFA.2008.4638461
18. Drey, Z., Faucher, C., Fleurey, F., Mahe, V., Vojtisek, D.: Kermeta language - Reference manual (2009). http://www.kermeta.org/docs/KerMeta-Manual.pdf
19. Eclipe Foundation: ATL Transformation Language (ATL) (2020). www.eclipse.org/atl/. Accessed 10 Jan 2020
20. Eclipe Foundation: Eclipse Modeling Framework (EMF) (2020). www.eclipse.org/modeling/emf/. Accessed on 10 Jan 2020
21. Eclipe Foundation: Eclipse Papyrus Modeling environment (2020). www.eclipse.org/papyrus/. Accessed on 10 Jan 2020
22. Eclipe Foundation: Visual Automated model TRansformations (VIATRA) (2020). www.eclipse.org/viatra/. Accessed on 10 Jan 2020
23. Eclipse Foundation: Xtend (2020). www.eclipse.org/xtend/. Accessed on 10 Jan 2020
24. Egyed, A., Zeman, K., Hehenberger, P., Demuth, A.: Maintaining consistency across engineering artifacts. Computer **51**(2), 28–35 (2018). https://doi.org/10.1109/MC.2018.1451666
25. Feldmann, S., Kernschmidt, K., Vogel-Heuser, B.: Combining a SysML-based Modeling Approach and Semantic Technologies for Analyzing Change Influences in Manufacturing Plant Models. Procedia CIRP **17**, 451–456 (2014). https://doi.org/10.1016/j.procir.2014.01.140
26. Fowler, M.: Domain-specific languages. Pearson Education, New York (2010)
27. Friedenthal, S., Moore, A., Steiner, R.: A practical guide to SysML: the systems modeling language. Morgan Kaufmann, Burlington (2014)
28. Hölldobler, K., Rumpe, B., Wortmann, A.: Software language engineering in the large: towards composing and deriving languages. Computer Lang., Syst. Struct. **54**, 386–405 (2018). https://doi.org/10.1016/j.cl.2018.08.002
29. Institute of Software Technology and Interactive Systems, Vienna University of Technology: Online Platform of the research project 'SysML4Industry' (2019). http://www.sysml4industry.org/?page_id=299. Accessed on 14 April 2019
30. International Electrotechnical Commission (IEC): Representation of process control engineering - Requests in P&I diagrams and data exchange between P&ID tools and PCE-CAE tools (2016)
31. International Electrotechnical Commission (IEC): Engineering Data Exchange Format for Use in Industrial Automation Systems Engineering – Automation Markup Language – Part 1: Architecture and General Requirements (2018)
32. International Electrotechnical Commission (IEC): Programmable controllers - Part 10: PLC open XML exchange format (2019)

33. International Organization for Standardization (ISO): Software engineering - Product quality - Part 1: Quality model (2001)
34. International Organization for Standardization (ISO): Industrial automation systems and integration - Product data representation and exchange (2018)
35. Johnson, T., Kerzhner, A., Paredis, C.J.J., Burkhart, R.: Integrating models and simulations of continuous dynamics into SysML. J. Comput. Inf. Sci. Eng. (2012). https://doi.org/10.1115/1.4005452
36. Jouault, F., Allilaire, F., Bézivin, J., Kurtev, I.: ATL: A model transformation tool. Sci. Computer Programm. **72**(1–2), 31–39 (2008). https://doi.org/10.1016/j.scico.2007.08.002
37. Kahani, N., Bagherzadeh, M., Cordy, J.R., Dingel, J., Varró, D.: Survey and classification of model transformation tools. Softw. Syst. Model. (2018). https://doi.org/10.1007/s10270-018-0665-6
38. Kernschmidt, K., Feldmann, S., Vogel-Heuser, B.: A model-based framework for increasing the interdisciplinary design of mechatronic production systems. J. Eng. Des. **29**(11), 617–643 (2018). https://doi.org/10.1080/09544828.2018.1520205
39. Kolahdouz Rahimi, S.: A comparative study of model transformation approaches through a systematic procedural framework and goal question metrics paradigm. Ph.D. thesis, King's College London (University of London) (2013). https://ethos.bl.uk/OrderDetails.do?uin=uk.bl.ethos.628396. Accessed on 10 March 2020
40. Krogstie, J.: Evaluating UML Using a Generic Quality Framework, p. 1-22. IGI Global (2003)
41. Lano, K.: The Agile UML Manual (2019). https://nms.kcl.ac.uk/kevin.lano/uml2web/umlrsds19.pdf
42. Lano, K., Kolahdouz-Rahimi, S.: Specification and verification of model transformations using UML-RSDS. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) **6396 LNCS**, 199–214 (2010). https://doi.org/10.1007/978-3-642-16265-7_15
43. Li, H., Zou, M., Hogrefe, G., Ryashentseva, D., Sollfrank, M., Koltun, G., Vogel-Heuser, B.: Application of a multi-disciplinary design approach in a mechatronic engineering toolchain. at - Automatisierungstechnik **67**(3), 246–269 (2019). https://doi.org/10.1515/auto-2018-0097
44. Modeling Languages Copyright: Executing ATL transformations from Java (2020). www.modeling-languages.com/executing-atl-transformations-java/. Accessed on 10 Jan 2020
45. Mohagheghi, P., Dehlen, V.: Developing a quality framework for model-driven engineering. In: Giese, H. (ed.) Models in software engineering, pp. 275–286. Springer, Berlin Heidelberg, Berlin Heidelberg (2008)
46. Object Management Group (OMG): UML Profile for MARTE Version 1.0 (2009). https://www.omg.org/spec/MARTE/1.0/
47. Object Management Group (OMG): Model Driven Architecture (MDA) – Guide 2.0 (2014). https://www.omg.org/cgi-bin/doc?ormsc/14-06-01
48. Object Management Group (OMG): Object Constraint Language (OCL) Specification Version 2.4 (2014). https://www.omg.org/spec/OCL/2.4
49. Object Management Group (OMG): Unified Modeling Language (UML) Specification Version 2.5 (2015). https://www.omg.org/spec/UML/2.5
50. Object Management Group (OMG): Meta object facility version 2.5.1 (2016). https://www.omg.org/spec/MOF/2.5.1
51. Object Management Group (OMG): Systems Modeling Language (SysML) Specification Version 1.6 (2017). https://www.omg.org/spec/SysML/1.6
52. Rose, L.M., Herrmannsdoerfer, M., Mazanek, S., Van Gorp, P., Buchwald, S., Horn, T., Kalnina, E., Koch, A., Lano, K., Schätz, B., Wimmer, M.: Graph and model transformation tools for model migration: Empirical results from the transformation tool contest. Softw. Syst. Model. **13**(1), 323–359 (2014). https://doi.org/10.1007/s10270-012-0245-0
53. Schmidt, D.: Guest editor's introduction: model-driven engineering. Computer **39**(2), 25–31 (2006). https://doi.org/10.1109/MC.2006.58
54. Shah, A.A., Kerzhner, A.A., Schaefer, D., Paredis, C.J.J.: Multi-view Modeling to Support Embedded Systems Engineering in SysML. In: Graph Transformations and Model-Driven Engineering, pp. 580–601. Springer Berlin Heidelberg (2010). https://doi.org/10.1007/978-3-642-17322-6_25
55. Steinberg, D., Budinsky, F., Paternostro, M., Merks, E.: EMF: Eclipse Modeling Framework. Eclipse Series. Addison-Wesley (2009) https://doi.org/10.5555/1197540. https://www.safaribooksonline.com/library/view/emf-eclipse-modeling/9780321331885/
56. Van Der Straeten, R., Mens, T., Van Baelen, S.: Challenges in model-driven software engineering. In: Models in Software Engineering, pp. 35–47. Springer Berlin Heidelberg (2009). https://doi.org/10.1007/978-3-642-01648-6_4
57. Van Solingen, R., Basili, V., Caldiera, G., Rombach, H.D.: Goal question metric (gqm) approach. Encycl. Softw. Eng. (2002). https://doi.org/10.1002/0471028959.sof142
58. Varró, D.: Incremental Queries and Transformations: From Concepts to Industrial Applications. pp. 51–59 (2016). https://doi.org/10.1007/978-3-662-49192-8_5
59. Varró, D., Bergmann, G., Hegedüs, Á., Horváth, Á., Ráth, I., Ujhelyi, Z.: Road to a reactive and incremental model transformation platform: three generations of the VIATRA framework. Softw. Syst. Model. **15**(3), 609–629 (2016). https://doi.org/10.1007/s10270-016-0530-4
60. Vogel-Heuser, B., Bougouffa, S., Sollfrank, M.: Researching evolution in industrial plant automation: Scenarios and documentation of the extended pick and place unit. Tech. Rep. TUM-AIS-TR-02-18-06, Technische Universität München (2018). https://mediatum.ub.tum.de/node?id=1468863. Accessed on 14 April 2019
61. Vogel-Heuser, B., Diedrich, C., Fay, A., Jeschke, S., Kowalewski, S., Wollschlaeger, M., Göhner, P.: Challenges for software engineering in automation. J. Softw. Eng. Appl. **07**(05), 440–451 (2014). https://doi.org/10.4236/jsea.2014.75041
62. Westfechtel, B.: A case study for a bidirectional transformation between heterogeneous metamodels in QVT relations. Commun. Computer Inf. Sci. **599**(1), 141–161 (2016). https://doi.org/10.1007/978-3-319-30243-0_8
63. Westfechtel, B.: Case-based exploration of bidirectional transformations in QVT relations. Softw. Syst. Model. **17**(3), 989–1029 (2018). https://doi.org/10.1007/s10270-016-0527-z
64. Whittle, J., Hutchinson, J., Rounce, M.: The state of practice in engineering. IEEE Softw. **31**(3), 79–85 (2014). https://doi.org/10.1109/MS.2013.65
65. Wohlin, C., Runeson, P., Höst, M., Ohlsson, M.C., Regnell, B., Wesslén, A.: Experimentation in software engineering. Springer Science and Business Media, New York (2012)

Gennadiy D. Koltun received a Dipl.-Ing. Degree in electrical engineering from Technical University of Dresden (TUD), Dresden, Germany, in 2016. He is currently pursuing a Ph.D. degree in mechanical engineering with the Institute of Automation and Information Systems at the Technical University of Munich (TUM). His main research interest is the model-based systems engineering of automated production systems.

Mathis Pundel received a B.Sc. (2017) and a M.Sc. (2020) degree in Mechanical Engineering from the Technical University of Munich (TUM). He is currently a solution developer of smart energy services at Nexxtlab. His main research interest is the development of model-driven engineering architectures for automated production systems.