The Syntax of a Multi-Level Production Process Modeling Language

Marko Vještica, Vladimir Dimitrieski, Slavica Kordić, Sonja Ristić, Ivan Luković
University of Novi Sad, Faculty of Technical Sciences,
Trg Dositeja Obradovića 6, 21102 Novi Sad, Serbia
Email: {marko.vjestica, dimitrieski, slavica, sdristic, ivan}@uns.ac.rs

Milan Pisarić
KEBA AG Linz, Gewerbepark Urfahr Reindlstraße 51,
4041 Linz, Austria
Email: pisa@keba.com

Abstract—**The fourth industrial revolution introduces changes in traditional manufacturing systems and creates basis for a lot-size-one production. The complexity of production processes is significantly increased, alongside the need to enable efficient process simulation, execution, monitoring, real-time decision making and control. The main goal of our research is to define a methodological approach and a software solution in which the Model-Driven Software Development (MDSD) principles and Domain-Specific Modeling Languages (DSMLs) are used to create a framework for the formal description and automatic execution of production processes. In that way production process models are used as central artefacts to manage the production. In this paper, we propose a DSML which can be used to create production process models that are suitable for automatic generation of executable code. The generated code is used for automatic execution of production processes within a simulation or a shop floor.**

I. Introduction

ADVANCED technologies in the form of smart resources and smart products are the basis for the fourth industrial revolution as they enable changes in factories and production. Industry 4.0 introduces primarily IT-driven changes in existing production systems in order to enable production of individualized products while preserving all beneficial economic characteristics of mass production [1].

Producing highly individualized products in traditional production facilities requires multiple production lines or, in case of a single production line, stopping the production to allow reconfiguration of machines which causes additional costs. To enable a flexible, individualized, lot-size-one production that is economically viable, the production needs to be carried out without stopping a production line for machine reconfiguration [2]. Therefore, it is necessary to solve the problem of tedious machine adaptation to frequent production changes that are common in the context of Industry 4.0. Additionally, there is a problem of frequent location changes of human workers in a factory [3]. Due to decreasing number of workers and increasing level of automation in factories, the workers are performing different tasks within a factory. Frequently changing worker's tasks increases production dynamics and requires fine coordination of workers in a factory so their work can be optimized, and production downtime avoided. As worker's tasks are often changed, a fast knowledge transfer is required so they do not lose time when changing workplaces.

To enable production of individualized products at the lower cost, a solution for production orchestration at a higher abstraction level can be utilized [4]. This solution would require a formal method to specify production processes and create process models that are suitable for automatic generation of instructions that are executed on smart resources. A smart resource represents a machine or a human worker that receives generated instructions and execute them on materials and products.

In this context, it is possible to apply a Model-Driven Software Development (MDSD) approach in which a centralized representation of knowledge would exist in a form of production process models. Therefore, in our previous work [5], we proposed a novel MDSD approach for production process modeling and automatic production process execution. The MDSD approach aims to reduce the gap between individual customer needs and the ability to produce required products. The main goals of the proposed MDSD approach are to: (i) enable easier adaptation of machines to dynamic changes of production processes, (ii) improve coordination of human workers and machines in factories and (iii) enable automatic execution of production processes. A formal specification of a production process is the crucial part of the proposed approach. Existing process modeling languages are not tailored to model production processes [6]. Currently, production processes are specified using different models like Bill of Materials (BOM), Flow Process Chart (FPC) and Failure Mode and Effects Analysis (FMEA) sheets. These models have different syntaxes and semantics. Therefore, it is hard to combine and reason production details from them in order to enable automatic execution of production processes.

To the best of our knowledge, there is no unified formal language aimed at modeling all production process aspects required for an automatic execution. Therefore, we decided to create a new Domain-Specific Modeling Language (DSML) aimed at production process modeling. Our MDSD

This paper is supported by KEBA AG Linz and by the Ministry of Education, Science and Technological Development through the project no. 451-03-68/2020-14/200156: "Innovative scientific and artistic research from the FTS domain".

approach, overviewed in Section 2, would enable flexible manufacturing with a help of Orchestrator software that manages production processes using a knowledge base and models created with the DSML. Orchestrator is a software running on a cluster of industrial computers that enables orchestration, detection and configuration of new and existing smart resources [7].

In this paper, we present abstract and concrete syntaxes of the DSML based on our previous research [5]. The Multi-Level Abstraction Approach (MLAA) is employed to develop the DSML. MLAA refers to representing objects at multiple levels of abstraction hierarchies. Due to the application of MLAA, we denote the language as Multi-level Production process modeling Language (MultiProLan). The higher level of abstraction enables easier production process modeling by specifying only production process steps, and the lower level of abstraction enables modeling of all the execution details dependent on a production system. MultiProLan allows process and quality engineers to collaborate on the specification of a production process by using a common language. In this paper, we denote process and quality engineers together as process designers. A process designer is a person in charge of transforming a valuable idea or experiment into an industrial process in a way to fulfil not only originality, efficiency, quality and sustainability criteria, but to consider a large number of often contradictory constraints.

MultiProLan enables modeling of production processes suitable for automatic execution. It can be used in a flexible and orchestrated production to facilitate the lot-size-one production. Supported with MultiProLan, our MDSD approach should increase the degree of factory automation by enabling easier adaptation of machines to dynamic production changes and by increasing coordination of resources in factories. Models expressed by the concepts of MultiProLan are simple enough for a human comprehension and can be also used as means of knowledge transfer to new workers or to workers that change their workplace frequently. Modeling production processes is important so human workers and supervisors could understand the processes better, eliminate potential modeling errors and optimize the processes.

Besides Introduction, this paper is structured as follows. An overview of the MDSD approach for modeling and automatic execution of production processes and the MultiProLan basic concepts are presented in Section 2. The related work that includes different modeling languages and approaches is summarized in Section 3. Abstract and concrete syntaxes of MultiProLan are described in Section 4. Conclusions and the future work are presented in Section 5.

II. An Overview of the MDSD Approach for Modeling and Automatic Execution of Production Processes

In the Model-Driven (MD) paradigm, models represent a central artefact at all stages of system development. A system developed by following the MD paradigm includes models that are connected and organized at different abstraction levels. An MDSD approach is a part of the MD paradigm and some of its goals are to: (i) increase software system developing speed through automatization and centralized representation of knowledge, (ii) increase software quality through formalization, (iii) increase reusability of models and (iv) lower system complexity through abstraction levels [8]. In MDSD approaches, DSMLs can be used and their purpose is to bring modeling concepts closer to users familiar with an application domain, so they can specify their solution with less time in comparison to General-Purpose Modeling Languages (GPMLs) [9]. Therefore, our opinion is that an MDSD approach and DSMLs will have significant role in enabling flexible, orchestrated and highly automated production. This is why we proposed a novel MDSD approach for modeling and automatic execution of production processes [5].

In Fig. 1, a system for automatic production orchestration and process execution is presented. The components that support the main steps of the MDSD approach are numerated and grouped within dashed rectangles. The proposed MDSD approach comprises the following steps: (i) specification of technological process models performed by process designers, (ii) automatic enrichment of technological process models with details needed for the execution, performed by Orchestrator on the basis of semantics gathered from Knowledge Base, (iii) generating the executable code performed by Code Generator and (iv) execution of generated instructions performed by Executor that forwards instruction to Digital Twin, and to the smart factory shop floor indirectly. A digital twin represents virtual model of a physical object. It can simulate the object behavior and the object can respond to changes made in the simulation [10]. The main part of the proposed system is MultiProLan created for the domain of hardware production. By using MultiProLan it is possible to create models that are suitable for automatic code generation and execution. The generated code represents human-readable or machine-

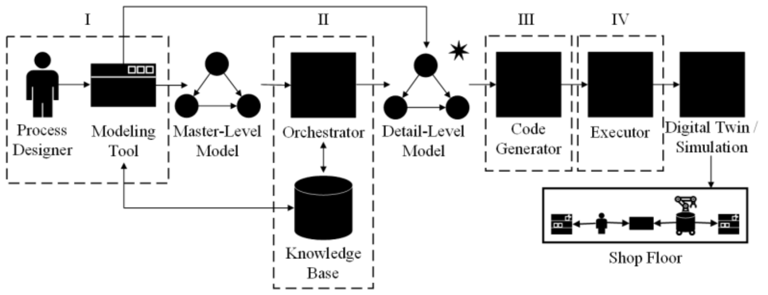

Fig. 1 The system for automatic production orchestration and process execution

readable instructions that are to be executed by smart resources. More detailed description of this approach is given in the rest of the section.

Specification of technological processes. The first step of the MDSD approach represents specification of production process models by using MultiProLan. These models include process steps without details required for automatic production, such as: smart resources required to execute process steps; production logistic activities; specific storages in which products and parts are stored; and machine configuration activities. A graphical modeling tool is implemented to allow the modeling of production processes using MultiProLan. Modeling Tool is used by process designers to model production processes at the higher level of abstraction. Such models are called Master-Level (ML) models. These models represent technological description of production processes and they include: (i) process steps, (ii) required capabilities, i.e. skills required to execute a process step, with their parameters and constraints (iii) input and output products, i.e. transformed resources like raw materials, components or finished goods, with constraints, (iv) workflows, i.e. sequence, parallelism, selection and iteration patterns, and (v) collaboration between process steps. A collaboration between smart resources, both humans and machines, is crucial in the context of Industry 4.0 [11] and it needs to be modeled. ML models do not depend on a specific technological platform, i.e. on a factory in which modeled production processes will be executed. Therefore, ML models can be considered as Platform-Independent Models (PIMs).

Enrichment of ML production process models with details needed for the execution. A production process will be executed within a given production system, e.g. some factory. To use an ML model for automatic code generation and execution, it is necessary to place additional information in it. This information refers to elements of a given production system. The information include: (i) specific resources like robots, machines and humans, that are to perform process steps, (ii) production logistic activities, which represent transportation of products and resources, and (iii) configuration of machines and robots like software setup, changing grippers, and plugging into a charger or a workstation. ML models enriched with aforementioned information are called Detail-Level (DL) models. DL models can be considered as Platform-Specific Models (PSMs) as they are enriched with details that are specific to a production system in which the models will be executed. The notions of ML and DL are introduced in this paper to better facilitate description of different modeling levels, and we did not come across them in surveyed literature.

DL models can be created manually or automatically. Manual DL creation is conducted by a process designer. A process designer can make additional changes to the existing ML/DL model or create a DL model from scratch using Modeling Tool. However, in our vision of the Industry 4.0 production process modeling, a production system and the production process models should be separated to enable a high level of a product customization. Thus, automatic creation of DL models is supported in our MDSD approach. The automatic DL creation from the existing ML model is conducted by the means of Orchestrator software. In the following text, automatic DL creation process is explained.

Knowledge Base needs to provide all the necessary information about a given production system for Orchestrator to be able to automatically generate DL models from ML models. Every process step specified in an ML model contains a capability, i.e. a skill that is required so that a process step can be executed. It is necessary to add the information about a resource that is to execute the process step within the given production system. This cannot be just any resource, but the resource that has the required capability in its set of offered capabilities. By using Knowledge Base, Orchestrator can match a capability that is required in a process step with a capability that a specific resource offers and, in that way, matches the process step with the resource. A capability of one process step could be matched with a capability of multiple resources. Orchestrator needs to use optimization techniques and scheduling mechanisms to choose one resource for every process step and to optimize work of resources in a factory. A process step that is ready to be executed is composed of: (i) input products, (ii) a capability needed to execute the process step, (iii) a smart resource that is to perform the capability on input products, and (iv) output products.

Orchestrator also needs to take care of production logistics. Orchestrator needs to add storages in which required products are stored and to add process steps that facilitate transportation of products and movement of resources between storages and workstations. Production logistic activities have a big impact on production processes as they require a lot of time [12], so it is very important to organize these activities well. Orchestrator also takes care of machine configurations. Based on knowledge gathered from Knowledge Base, Orchestrator can infer whether the machine configuration step needs to be added to the process to enable further activities.

For Orchestrator to be able to reach the aforementioned conclusions, Knowledge Base needs to contain knowledge of production system elements, such as: (i) smart resources with their set of capabilities, (ii) smart products with their attributes like dimensions and weight (iii) process steps with required products and capabilities, (iv) production logistics and (v) configuration process steps that are required by some resources prior or after execution of another process step. In this paper, we look at Orchestrator as a black box. It is presented just to provide context in which MultiProLan is used. An internal structure of Orchestrator that is used in the MDSD approach can be found in our previous work [7].

An ML model exists independently from a production system that will be the execution platform for the modeled production process. At this high abstraction level, a process designer does not need to take care for the specific details of a given production system. These details must be specified within Knowledge Base before the specification of a DL model begins. DL models can contain only those capability, product, resource, storage, constraint and parameter details that are already specified in Knowledge Base. In that context, a DL model is specified whenever an execution-ready production process model is needed, and it is dependent on a production system.

Generating the executable code. The third step of the MDSD approach represents code generation from DL models. It is possible to send DL models into Code Generator so it could automatically generate instructions that can be executed by human workers or machines. More details on Code Generator can be found in [7].

Execution of generated code instructions. Executor forwards generated instructions to Digital Twin, which represents both simulation and command proxies to the shop floor. In our case, the Digital Twin component could be used for the simulation only or it could also forward instruction to shop floor smart resources through embedded proxies and mobile devices [7]. Using a digital twin in the simulation-only mode could decrease production failures, provide insight into badly modeled process steps and enable optimization of resources and processes [13]. By running simulations it is possible to predict an influence of process steps to a final product [14].

III. Related Work

Production processes should be digitally supported in Industry 4.0 [15] so they can be integrated within a smart factory. Modeling production processes is very important in industrial informatics [16], but it is not enough to document processes and store them in a factory database. Production processes should be modeled to lead the production. Process models should be ready for automatic production, but also not too complex for a human comprehension. In this section, different production process modeling approaches and languages are presented, as well as their capabilities to fulfill the aforementioned needs.

Companies mostly use manufacturing process charts and BOMs to specify production processes, but none of these specifications provide enough data to facilitate automatic execution. BOM specifications are not enough to understand a production flow [17]. On the other hand, Bill of Materials and Operations (BOMO) [18] specifications cover the production flow, but are insufficient to specify selection and iteration patterns or smart resources. There is also Korean manufacturing process chart standard KS A 3002 [19], but a tooling support and a possibility to automatically execute models are missing [17]. Unified Modeling Language (UML) activity diagrams are used to describe production processes, but models are not suitable for the automatic execution, they are not intuitive for process designers and they could be complex [20].

By using conceptual process modeling languages like UML activity diagram, Business Process Modeling and Notation (BPMN) and Petri nets, it is difficult to model production processes primarily as they are not created for that purpose. These difficulties are even more noticeable whenever the languages need to cover all production process concepts required for the automatic execution [6]. To solve this problem researches usually extend existing languages to add missing semantics. However, these extensions are not enough to solve the problem due to the wide application domain of a language. Therefore, researches often try to create new domain-specific languages instead of extending existing general-purpose languages [21].

Zor et al. proposed BPMN extensions to model production processes [22], however it is difficult to model a material flow [23] and the whole context of production domain is not covered due to the absence of uniformity [17]. BPMN extensions are also proposed by Ahn and Chang for production process similarity measurements [17], however it is not possible to model selection and iteration patterns or to specify smart resources. According to Lütjen and Rippel [23], some languages like DELMIA Process Engineer, Systems Modeling Language (SysML) and Petri nets lack in possibility to specify the material flow. To overcome the usual lack of the material flow modeling concept, the same authors proposed a novel material flow-oriented process modeling language – GRAMOSA, but the material flow-oriented approach was complex [23].

Meyer et al. [24] proposed BPMN extensions to model Internet of Things (IoT) devices and create IoT-aware process models. Besides humans that participate in business process executions, IoT-aware processes also include IoT devices that can do some of tasks in a smart factory. Likewise, Petrasch and Hentschke [25] proposed IoT-Aware Process Modeling Method (IAPMM) using UML use cases and BPMN extensions in order to model IoT-aware processes. The goal of this method is to enable modeling of software systems and software applications like sensing and actuation. The same authors extended IAPMM and created Industry 4.0 Process Modeling Language (I4PML) [26] by adding extensions like Cloud Computing applications. Using this language, it is not possible to model all the technological details, as its purpose is to model production processes in a requirements specification and analysis phase. According to Schönig et al. [27], none of the aforementioned languages and approaches provides details on how to execute models. This is the reason why they proposed an approach for integration of IoT objects with business process models ready for an execution. They extended BPMN to enable integration of IoT objects with process models, but also to preserve a possibility to execute the models in existing Business Process Management (BPM) execution systems. However, it would be difficult to specify the material flow, smart resources, products, capabilities and constraints, and thus the full automatization, in which both humans and machines participate, would be hard to achieve. Because of these insufficiencies, Orchestrator would not be able to manage the production based on the models.

Witsch and Vogel-Heuser [20] presented Manufacturing Execution System Modeling Language (MES-ML) whose purpose is to specify MES through different views so that model complexity could be reduced. MES-ML is based on BPMN and covers the modeling of a technical system, production processes and MES/IT functions. By using links, it is possible to connect process steps with production system elements, i.e. smart resources, that will execute the steps. This way a dependency between production process models and a production system is created. Due to this dependency, a process designer needs to take care how to connect process steps with production system elements during the production process modeling. This makes the production process modeling significantly more difficult and could lead to higher number of created errors during the modeling and higher model complexity.

According to Weissenberger et al. [28], MES-ML does not support creation of generic production processes as the semantics of process tasks are insufficiently specified and process models are not suitable for code generation. To enable the modeling of machine-usable MES specifications suitable for code generation, the same authors implemented a DSML by extending MES-ML. The goal of this language is to enable higher independency of production process models from a production system during process modeling. Instead of the link that is used to connect a process step with a resource of a production system, the authors proposed a list of links to be used. At the runtime, resources that execute process steps will be determined. However, the dependency between process steps and production system resources still exists and it is ambiguous which resources will execute process steps until the runtime.

Similar to the previous work, Fallah et al. [4] presented a framework to model a modular MES using SysML. However, the framework is not implemented. Neither a code generator for model transformation into executable code nor an interpreter for direct model execution are implemented.

Because of the dependency between production process models and a production system, we decided to create the language with two levels of abstraction. In this way, process designers do not need to take care of production system elements during the production process modeling and they can be entirely focused on modeling process steps. Production process models become more generic by separating a production system from them. It is possible to automatically connect process steps with smart resources in the runtime without additional load to process designer by using Orchestrator. As we could not find any formal language that allows creation of generic production process models suitable for automatic execution, we decided to create a novel DSML. This DSML unifies all production process aspects, as mentioned in Section 1, and thus enables the specification of DL models that are used for automatic code generation and production process execution. ML models are separated from a production system so that process designers could model them in more generic way.

IV. Abstract and Concrete Syntaxes of MultiProLan

In this section we present abstract and concrete syntaxes of MultiProLan for modeling production processes suitable for automatic code generation and execution. We use an Ecore meta-meta-model, which is a part of Eclipse Modeling Framework (EMF) [29], to create the abstract syntax of MultiProLan. Also, we use the Eclipse Sirius framework [30] to create the graphical concrete syntax and to enable simple implementation of a prototype tool.

A. The Abstract Syntax of MultiProLan

Two levels of abstraction are needed to ease the modeling performed by process designers, but also to fully prepare models for an execution phase. A higher abstraction level – ML separates production process models from a production system, while a lower abstraction level – DL enables creation of production process models that are executable within a given production system. Based on these levels of abstraction, we divided the meta-model into two parts. This was also done because the meta-model is more concise and easier to understand.

The ML part of the MultiProLan meta-model is depicted in Fig. 2 and it represents production process modeling concepts needed at the higher level of abstraction. These concepts are used by process designers to create ML models. A production process is modeled by the *Process* class which represents the root model element. A process version must be specified as models are stored in a knowledge base and can be changed or reused at any time. A process is composed of process elements (*ProcessElement*), which can

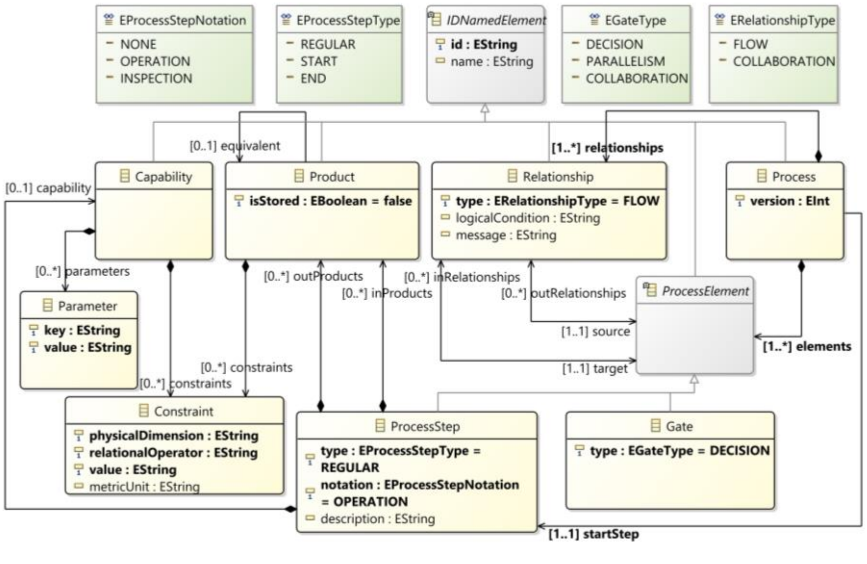

Fig. 2 The first part of the meta-model used for ML model creation

be process steps (*ProcessStep*) or gates (*Gate*), and relationships (*Relationship)* between them. The start process step must be referenced from a process (*startStep*) as knowledge of the execution starting point is needed. There are two types of relationships (*ERelationshipType)*: (i) flow – representing a workflow between process elements, and (ii) collaboration – representing a message flow between process steps. Relationships have the message attribute specified whenever a message needs to be sent between collaboration process steps. Also, relationships have the logical condition specified whenever they are used in selection or iteration patterns.

A process step is composed of a capability (*Capability*) and products (*Product*) on which the capability is to be performed. Input products (*inProducts*) represent products on which a capability is performed, i.e. raw materials, and output products (*outProducts*) represent products that are the result of the capability usage, i.e. finished goods. Process steps can be of different types (*EProcessStepType*): (i) start – the first process step, (ii) end – the last process step or (iii) regular – other process steps that contain capabilities that must be performed on products. Start and end process steps do not have any capability or product, and only one start process step and only one end process step have to exist per each production process model. A process step has a notation (*EProcessStepNotation*) which has one of the following values: (i) none – for start and end process steps, (ii) operation – an activity that changes input products and creates output products and (iii) inspection – an activity to check quality of products.

A material flow should be specified for every product. An input product can be equivalent (*equivalent*) to an output product of the previous process step, or it can be brought from a storage. An output product can be used in following process steps or it can be stored in a storage. Every product and capability have constraints (*Constraint*) such are dimensions, color and weight that will be considered by Orchestrator when it decides which smart resource is able to perform a process step. Some capabilities require parameters (*Parameter*) to be specified, e.g. to drill a hole, the drilling position must be specified.

Besides process steps, there are also gates that are used as process elements. Gates are elements that are needed in order to create: (i) selection and iteration patterns – flow control in processes, (ii) parallelism – two or more process steps need to be executed in parallel and (iii) collaboration – two or more process steps need to be executed in parallel, but one process step must not start or finish its activity before gets a message that another process step finished its activity. Finally, most of the presented classes inherit the *IDNamedElement* class comprising *id* and *name* attributes.

The DL part of the MultiProLan meta-model is depicted in Fig. 3 and it represents production process modeling concepts needed at the lower level of abstraction. This part of the meta-model is an extension of the ML part and together they are used to create DL models. Process step notations are extended by (i) transportation – production logistic activities, (ii) configuration – activities to configure resources and (iii) delay – necessary waiting activities. A process step is extended with a resource that will execute it by using a required capability. A resource (*Resource*) can be an actuator – an active resource, i.e. one that performs different activities during the production, or a storage – a passive resource, i.e. one that stores products. A resource can be both an actuator and a storage, e.g. there are robots that can execute different tasks, but also have a place to temporarily store products. A resource can be a human worker or a machine (*EResourceType*) and it can also represent an actuator or a storage. Depending on the resource type, human-readable or machine-readable instructions will be generated for every process step. Also, a resource could be of type *NONE* which means that it is neither a human nor a machine, e.g. a regular storage shelf, with no smart devices or sensors attached. Products are extended with a specific storage that must be defined for every input product brought from the storage and for every output product placed in the storage. When extended with active and passive resources, production logistics and configuration activities, process steps are ready for the automatic code generation and execution.

B. *The Concrete Syntax of MultiProLan*

There are two types of concrete syntaxes – textual and graphical, but there is no general answer which one is more suitable [31]. We decided to create the graphical syntax for MultiProLan to make the modeling easier for production

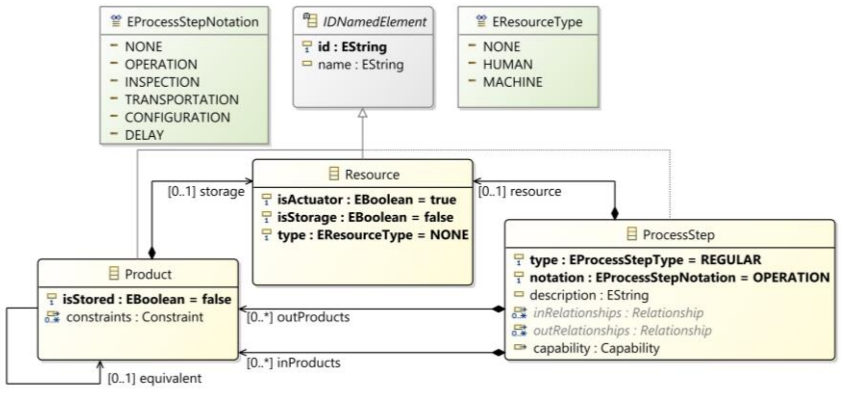

Fig. 3 The second part of the meta-model used for DL model creation

process designers as they are already familiar with other graphical languages, such as FPC. The decision was also made to enable visualized process monitoring, as well as to enable visualization of detected errors during the production. As BPMN [32] is commonly used to model different kind of processes and as it is easy to interpret its models [33], some BPMN concepts, such as activities and gates, are used in the graphical syntax of MultiProLan. The graphical syntax is also inspired by American Society of Mechanical Engineers (ASME) FPCs [34] as process designers are used to these charts. Some of FPC elements are used in process step notations, such as: operation, transportation, inspection, and delay. Also, the storage element is used within a product, indicating that a product should be gathered from a storage or placed in a storage. The symbols used for the MultiProLan concrete syntax are presented in Fig. 4.

The concrete syntax is described within production process model examples presented in Fig. 5 and Fig. 6. These two examples represent a process of a wooden box production at ML and DL of abstraction, respectively. The box is composed of four wooden planks that represent different sides of the box, and of a thin wooden back side. The four wooden planks can be assembled into a frame using wooden pins, and the wooden back side needs to be hammered into the frame, creating the box. The production of the wooden box is installed in a smart factory composed of: (i) the smart shelf – storage in which wooden planks are stored, (ii) the first assembly table – storage that is used to assemble four wooden sides, (iii) the second assembly table – storage that is used to hammer the back side into the frame, (iv) the recycle bin – storage for impaired boxes, (v) the finishing area – storage for finished boxes and (vi) human workers and industrial mobile robots – smart resources that are able to perform required activities.

The ML model of the wooden box production is presented in Fig. 5. The presented ML model is composed of six parts: (i) the *start* process step, (ii) parallel process steps of assembling left-bottom and right-upper sides, after which these two assembled sides should also be assembled into the frame, (iii) collaboration process steps of holding the frame and hammering the back side into the frame, (iv) inspection of the box, (v) decision whether the box needs to be stored or discarded, depending on results of the *inspection* process step and (vi) the *end* process step. The process step of assembling the left-bottom side represents an operation as it is depicted with a circle icon at the left side of the process step name. It has two input products, left and bottom sides of the frame, both gathered from a storage. The inverted triangle icon at the left side of a product name represents that an input product should be gathered from a storage, or that an output product should be placed in a storage. Two input products have two constraints, width and height, that will be considered by Orchestrator when it assigns a smart resource that is able to pick the plank of these dimensions. The same process step has the *assemble* capability with parameters that represent two wooden pins with the space between them of 0.07m. The output product of this process step is the assembled left-bottom side, which will not be stored, but will be used by the next process step. Assembling the right-upper side is an equivalent process step to assembling the left-bottom side process step. Both process steps need to be executed in parallel, as they are modeled between two parallelism gates (*PAR*). The next process step requires to assemble the frame and it has two input products, which are left-bottom and right-upper sides from the previous two process steps. These input products are not gathered from a storage but are equivalent to the previous process steps output products, as it is depicted by directed dashed lines in the process diagram. This process step has the *assemble* capability and the frame as the output product. The same frame is held in the next process step. This process step is a part of the collaboration activities, represented between two

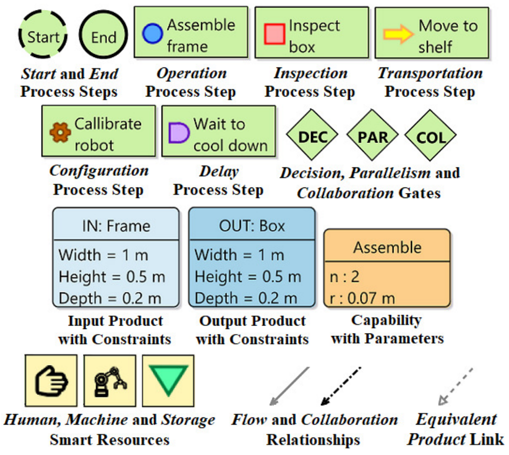
Start
End
Assemble frame
Inspect box
Move to shelf
Start and End Process Steps
Operation Process Step
Inspection Process Step
Transportation Process Step
Callibrate robot
Wait to cool down
DEC
PAR
COL
Configuration Process Step
Delay Process Step
Decision, Parallelism and Collaboration Gates
IN: Frame
Width = 1 m
Height = 0.5 m
Depth = 0.2 m
OUT: Box
Width = 1 m
Height = 0.5 m
Depth = 0.2 m
Assemble
n : 2
r : 0.07 m
Input Product with Constraints
Output Product with Constraints
Capability with Parameters
Human, Machine and Storage Smart Resources
Flow and Collaboration Relationships
Equivalent Product Link

Fig. 4 Symbols of the MultiProLan concrete syntax

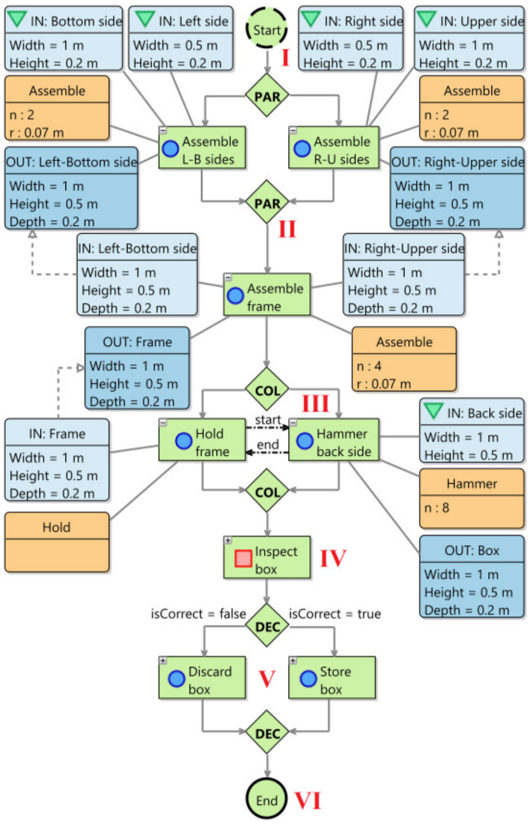

Fig. 5 The ML model of the wooden box production example

collaboration gates (*COL*). It does not have an output product as it is the same as the input product. Another process step of the collaboration activities is to hammer the back side into the frame that is held. Hammering the back side should not start before the message arrives that the frame is being held. The frame should be held until the message arrives that the hammering is finished. This is presented in the process diagram with dotted-line relationships between those two process steps. The input product of the hammering process step is the back side that should be gathered from a storage and the output product is the box. The *hammer* capability has predefined number of nails that should be hammered, e.g. eight, and after the hammering is finished, the message is sent to the *hold* process step. After the collaboration process steps are finished, the box is inspected for any deformation. The *inspection* process step and process steps between decision gates also have input and output products and a capability, but they are hidden from the diagram using the +/- button at the top left corner of process steps. The decision of storing or discarding the box should be made depending on whether the box passes all checks. These process steps are modeled between two decision gates (*DEC*). The process is finished after it reaches the *end* process step.

Based on the presented ML model and knowledge from Knowledge Base, Orchestrator generates the DL model of the wooden box production, which is presented in Fig. 6. Due to the paper length limitations, products and capabilities are depicted just for process steps in the left parallelism branch, while for other process steps they are modeled, but not presented on the diagram. Like the presented ML model, the generated DL model is composed of the same six parts, but the model is extended with additional details and new process steps, like production logistic activities and mobile robot configurations. These new process steps are needed to automatically produce the box. In the rest of this subsection, we describe some of the process steps, while others are extended in the similar way. The *assemble left-bottom side* and the *assemble right-upper side* process steps are assigned in parallel to a human worker and an industrial mobile robot, respectively. In both parallel branches transportation process steps have been added, which are depicted with the arrow icon at the left side of the process step name. To assemble the left-bottom side, the human worker needs to move to the smart shelf, pick left and bottom sides, move to the first assembly table and assemble these two sides. Transportation process steps only have the *move* capability with the *location* parameter, as products for these steps do not exist. The *pick* process steps have a capability and an input product, but an output product does not exist. Unlike the ML model in which input products have general storages as an indicator that they need to be gathered, the DL model input products have the specific storages, e.g. smart shelf, from which the products need to be gathered. These specific storages are depicted by inverse triangle objects set on input products. By selecting a storage, it is possible to specify values of the storage attributes, but this is not presented in the diagram due to the paper length limitations. Similar could be done with resources set on process steps. As for the *assemble* process step input products, they are equivalent to previously picked products, which is denoted with the directed dashed lines between equivalent products. The capability and the output product of this process step are the same as in the ML model.

Another parallel branch represents assembling of the right-upper side by the industrial mobile robot. Process steps in this branch are similar to process steps of the previously described branch, except of the configuration process steps. As the industrial mobile robot assigned to these process steps is not equipped with the machine vision modules, therefore it must be calibrated after each movement to

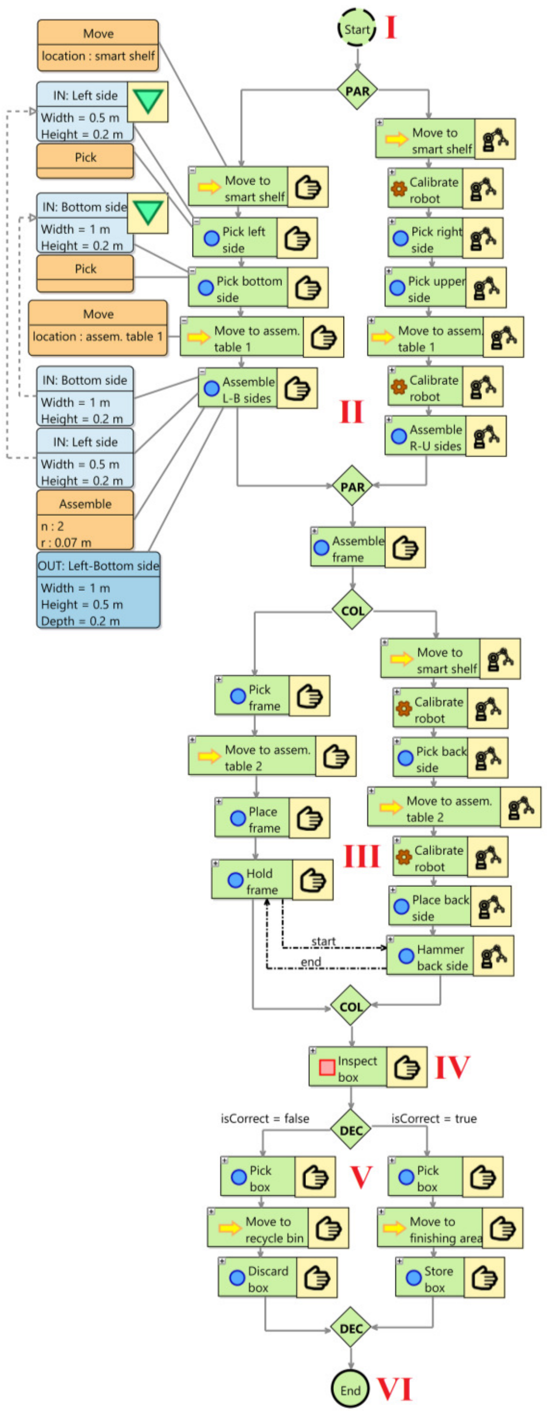

Fig. 6 The DL model of the wooden box production example

determine its position. Configuration process steps can be differentiated from other process steps by the gear icon at the left side of the process step name. After the left-bottom and right-upper sides are assembled, the same human worker needs to assemble the frame. This activity does not require any transportation process steps as the human worker and the required input products are already at the first assembly table. The assembled frame is used in the collaboration process steps that are extended with transportation and configuration process steps in the similar way. The frame should be transported to the second assembly table and the back side should be gathered from the smart shelf and transported to the same table. Hammering the back side into the frame should not start before the frame is transported and placed on the second assembly table and is being held. Also, holding the frame should not end until the hammering is finished, and the box is produced. The human worker then visually inspects the box for any deformations. Via a mobile device the human worker gets detailed instructions generated from the *description* attribute and checks whether the box passes the inspection. The decision must be made whether the box should be transported and discarded into the recycle bin or should be transported and stored into the finishing area. Any of these two cases will be done by the human worker.

The presented DL model is suitable for automatic execution. Code Generator will generate instructions from the DL model and Executor will send the instructions to smart resources and wait for their response. After the response arrives, Executor will send subsequent instructions until the production is finished. Code Generator generates generic instructions that are passed to Digital Twin in an appropriate protocol. Digital Twin receives and transforms messages into human-readable or machine-specific commands and passes them for execution. Digital Twin also updates the digital footprint of all resources it contains.

V. Conclusion and Future Work

In this paper we presented the DSML for modeling hardware production processes suitable for automatic execution. The goal of the language is to support the modeling of all production details required for automatic execution, but not to be too complex for a human to comprehend. To achieve this goal, two levels of abstraction are implemented so that production processes could be modeled in a generic way. By creating two levels of abstraction, production process models become independent from the production system details and thus efforts needed during the production process modeling are reduced. According to our experience from the industry, a process designer still needs to have the knowledge about the production system. Consequently, it is hard to make strict separation between production process models at PIM and PSM levels. However, we aim to achieve this separation by creating ML models and automatically generating DL models from them by using Orchestrator and the domain knowledge represented in a machine-readable way. Thus, the presented research leads one step closer to this goal. The language also allows process and quality engineers to collaborate on the creation of production process models. Created models could be used as a central artefact in a smart factory and thus lead the production automatically. Such language is implemented in a formal way and thus should increase consistency during modeling and decrease the amount of time needed for modeling. Integrating the language within the proposed MDSD approach should increase the production flexibility and contribute to the faster lot-size-one production.

One of the key future steps of our research will be to conduct the evaluation of the presented language. Using Modeling Tool, the language is tested by industrial process designers within an industrial use case [35], but we plan to systematically conduct the language evaluation that will include researchers and students from the academic community and process designers from the industry. During the initial MultiProLan validation, process designers were able to easily model the entire production process they needed and send the models to Orchestrator for execution. The evaluation should verify whether the language with multiple abstraction levels could make the modeling of production processes suitable for automatic execution easier comparing with other languages and approaches. Also, the evaluation should verify whether the language contributes to increasing the factory automation degree.

We will expand the language with concepts of quality assurance and error handling, as an occurrence of any failure requires error handling that needs to be carefully carried out and modeled [2]. Modeling production errors will cover all the basic attributes of FMEA documentation as the FMEA sheets will be automatically generated from process models. Also, an automatic generation of user manuals is needed. These documents contain a textual description of every process step and images on how to execute these steps. Currently, our Code Generator only generates human-readable or machine-readable instructions for the automatic process execution and should be extended with a feature to generate FMEA sheets, user manuals, BOMs and FPCs.

In addition to the error modeling, we plan to extend the language with: (i) subprocesses – to lower complexity of graphical process models, (ii) unordered process steps – as some activities could be executed in any order, e.g. in Fig. 6, the *pick left side* and the *pick bottom side* process steps should be unordered process steps and (iii) process variations – when the same result could be done by executing different process steps.

As the language is currently designed to model a hardware production, it could be extended to support the modeling of: (i) process production, e.g. breweries, sugar factories, pharma factories, (ii) software production and (iii) provision of service processes, e.g. banks, health care, education. Also, currently there is only the graphical syntax of the language. A textual syntax should also be implemented as some process designers could find it easier to use than the graphical syntax, or they could use the combination of these two syntaxes.

As a part of the future work, we also plan to further investigate the usability of MultiProLan. The emphasis will be on the collaboration between various participants and artefacts during the specification of a production process model.

REFERENCES

[1] H. Lasi, P. Fettke, H.-G. Kemper, T. Feld, and M. Hoffmann, "Industry 4.0," *Bus. Inf. Syst. Eng.*, vol. 6, no. 4, pp. 239–242, Aug. 2014, doi: https://doi.org/10.1007/s12599-014-0334-4.

[2] K. Dorofeev, S. Profanter, J. Cabral, P. Ferreira, and A. Zoitl, "Agile Operational Behavior for the Control-Level Devices in Plug&Produce Production Environments," in *Proceedings of 24th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, Zaragoza, Spain, 2019, pp. 49–56, doi: https://doi.org/10.1109/ETFA.2019.8869208.

[3] D. Gorecky, M. Schmitt, M. Loskyll, and D. Zuhlke, "Human-machine-interaction in the industry 4.0 era," in *Proceedings of 2014 12th IEEE International Conference on Industrial Informatics (INDIN)*, Porto Alegre RS, Brazil, Jul. 2014, pp. 289–294, doi: https://doi.org/10.1109/INDIN.2014.6945523.

[4] S. M. Fallah, S. Wolny, and M. Wimmer, "Towards model-integrated service-oriented manufacturing execution system," in *2016 1st International Workshop on Cyber-Physical Production Systems (CPPS)*, Vienna, Austria, Apr. 2016, pp. 1–5, doi: https://doi.org/10.1109/CPPS.2016.7483917.

[5] M. Vještica, V. Dimitrieski, M. Pisarić, S. Kordić, S. Ristić, and I. Luković, "Towards a formal description and automatic execution of production processes," in *Proceedings of 2019 IEEE 15th International Scientific Conference on Informatics*, Poprad, Slovakia, Nov. 2019, pp. 463–468, doi: https://doi.org/10.1109/Informatics47936.2019.9119314.

[6] M. Vještica, V. Dimitrieski, M. Pisarić, S. Kordić, S. Ristić, and I. Luković, "Towards a Formal Specification of Production Processes Suitable for Automatic Execution," *Open Comput. Sci.*, p. 20, May 2020, to be published.

[7] M. Pisarić, V. Dimitrieski, M. Vještica, and G. Krajoski, "Towards a Non-Disruptive System for Dynamic Orchestration of the Shop Floor," in *IFIP Advances in Information and Communication Technology (AICT)*, Novi Sad, Serbia, 2020, vol. 592, pp. 1–8, doi: https://doi.org/10.1007/978-3-030-57997-5_54.

[8] V. Dimitrieski, "Model-Driven Technical Space Integration Based on a Mapping Approach," Ph.D. Thesis, University of Novi Sad, Faculty of Technical Sciences, Serbia, 2017.

[9] M. Mernik, J. Heering, and A. M. Sloane, "When and how to develop domain-specific languages," *ACM Comput. Surv.*, vol. 37, no. 4, pp. 316–344, Dec. 2005, doi: https://doi.org/10.1145/1118890.1118892.

[10] Q. Qi and F. Tao, "Digital Twin and Big Data Towards Smart Manufacturing and Industry 4.0: 360 Degree Comparison," *IEEE Access*, vol. 6, pp. 3585–3593, 2018, doi: https://doi.org/10.1109/ACCESS.2018.2793265.

[11] C. Leyh, S. Martin, and T. Schäffer, "Industry 4.0 and Lean Production – A Matching Relationship? An analysis of selected Industry 4.0 models," in *Proceedings of 2017 Federated Conference on Computer Science and Information Systems (FedCSIS)*, Sep. 2017, vol. 11, pp. 989–993, doi: https://doi.org/10.15439/2017F365.

[12] T. Qu, S. P. Lei, Z. Z. Wang, D. X. Nie, X. Chen, and G. Q. Huang, "IoT-based real-time production logistics synchronization system under smart cloud manufacturing," *Int. J. Adv. Manuf. Technol.*, vol. 84, no. 1–4, pp. 147–164, Apr. 2016, doi: https://doi.org/10.1007/s00170-015-7220-1.

[13] S. Vaidya, P. Ambad, and S. Bhosle, "Industry 4.0 – A Glimpse," in *Procedia Manufacturing*, Maharashtra, India, 2018, vol. 20, pp. 233–238, doi: https://doi.org/10.1016/j.promfg.2018.02.034.

[14] J. Wan, H. Cai, and K. Zhou, "Industrie 4.0: Enabling Technologies," in *Proceedings of 2015 International Conference on Intelligent Computing and Internet of Things*, Harbin, 2015, pp. 135–140, doi: https://doi.org/10.1109/ICAIOT.2015.7111555.

[15] L. D. Xu, E. L. Xu, and L. Li, "Industry 4.0: state of the art and future trends," *Int. J. Prod. Res.*, vol. 56, no. 8, pp. 2941–2962, Apr. 2018, doi: https://doi.org/10.1080/00207543.2018.1444806.

[16] L. D. Xu, "Enterprise Systems: State-of-the-Art and Future Trends," *IEEE Trans. Ind. Inform.*, vol. 7, no. 4, pp. 630–640, Nov. 2011, doi: https://doi.org/10.1109/TII.2011.2167156.

[17] H. Ahn and T.-W. Chang, "Measuring Similarity for Manufacturing Process Models," in *IFIP Advances in Information and Communication Technology (AICT)*, Cham, Aug. 2018, vol. 536, pp. 223–231, doi: https://doi.org/10.1007/978-3-319-99707-0_28.

[18] J. Jiao, M. M. Tseng, Q. Ma, and Y. Zou, "Generic Bill-of-Materials-and-Operations for High-Variety Production Management," *Concurr. Eng.*, vol. 8, no. 4, pp. 297–321, Dec. 2000, doi: https://doi.org/10.1177/1063293X0000800404.

[19] Korean Standards Service Network (KSSN), "KS A 3002 Standard." https://www.kssn.net/en/ (accessed Apr. 05, 2020).

[20] M. Witsch and B. Vogel-Heuser, "Towards a Formal Specification Framework for Manufacturing Execution Systems," *IEEE Trans. Ind. Inform.*, vol. 8, no. 2, pp. 311–320, May 2012, doi: https://doi.org/10.1109/TII.2012.2186585.

[21] A. Wortmann, O. Barais, B. Combemale, and M. Wimmer, "Modeling Languages in Industry 4.0: An Extended Systematic Mapping Study," *Softw. Syst. Model.*, vol. 19, pp. 67–94, Jan. 2020, doi: https://doi.org/10.1007/s10270-019-00757-6.

[22] S. Zor, D. Schumm, and F. Leymann, "A Proposal of BPMN Extensions for the Manufacturing Domain," in *Proceedings of the 44th CIRP International Conference on Manufacturing Systems*, Madison, Wisconsin, USA, 2011, pp. 1–7.

[23] M. Lütjen and D. Rippel, "GRAMOSA framework for graphical modelling and simulation-based analysis of complex production processes," *Int. J. Adv. Manuf. Technol.*, vol. 81, no. 1–4, pp. 171–181, May 2015, doi: https://doi.org/10.1007/s00170-015-7037-y.

[24] S. Meyer, A. Ruppen, and L. Hilty, "The Things of the Internet of Things in BPMN," in *Advanced Information Systems Engineering Workshops. CAiSE 2015. Lecture Notes in Business Information Processing*, Stockholm, Sweden, 2015, vol. 215, pp. 285–297, doi: https://doi.org/10.1007/978-3-319-19243-7_27.

[25] R. Petrasch and R. Hentschke, "Towards an Internet-of-Things-aware Process Modeling Method - An Example for a House Suveillance System Process Model," in *Proceedings of 2nd Management and Innovation Technology International Conference (MITiCON2015)*, Bangkok, Thailand, 2015, pp. 168–172.

[26] R. Petrasch and R. Hentschke, "Process modeling for industry 4.0 applications: Towards an industry 4.0 process modeling language and method," in *Proceedings of 2016 13th International Joint Conference on Computer Science and Software Engineering (JCSSE)*, Khon Kaen, Thailand, Jul. 2016, pp. 1–5, doi: https://doi.org/10.1109/JCSSE.2016.7748885.

[27] S. Schönig, L. Ackermann, S. Jablonski, and A. Ermer, "IoT meets BPM: a bidirectional communication architecture for IoT-aware process execution," *Softw. Syst. Model.*, Mar. 2020, doi: https://doi.org/10.1007/s10270-020-00785-7.

[28] B. Weissenberger, S. Flad, X. Chen, S. Rosch, T. Voigt, and B. Vogel-Heuser, "Model driven engineering of manufacturing execution systems using a formal specification," in *Proceedings of 2015 IEEE 20th Conference on Emerging Technologies & Factory Automation (ETFA)*, Luxembourg, Sep. 2015, pp. 1–8, doi: https://doi.org/10.1109/ETFA.2015.7301430.

[29] D. Steinberg, F. Budinsky, M. Paternostro, and E. Merks, *EMF: Eclipse Modeling Framework*, 2nd ed. Upper Saddle River, NJ, USA: Addison-Wesley Professional, 2008.

[30] "Eclipse Sirius Documentation." https://www.eclipse.org/sirius/doc/ (accessed Mar. 19, 2020).

[31] I. Dejanovic, M. Tumbas, G. Milosavljevic, and B. Perisic, "Comparison of Textual and Visual Notations of DOMMLite Domain-Specific Language," in *Local Proceedings of the Fourteenth East-European Conference on Advances in Databases and Information Systems*, Novi Sad, Serbia, Sep. 2010, pp. 131–136.

[32] Object Management Group, "Business Process Model and Notation, Version 2.0.2," Technical Report, 2014.

[33] M. Kocbek, G. Jost, M. Hericko, and G. Polancic, "Business process model and notation: The current state of affairs," *Comput. Sci. Inf. Syst.*, vol. 12, no. 2, pp. 509–539, 2015, doi: https://doi.org/10.2298/CSIS140610006K.

[34] American Society of Mechanical Engineers. Special committee on standardization of therbligs, process charts, and their symbols, *A.S.M.E. standard operation and flow process charts*. New York, N.Y., The American society of mechanical engineers, 1947.

[35] M. Vještica, V. Dimitrieski, M. Pisarić, S. Kordić, S. Ristić, and I. Luković, "An Application of a DSML in Industry 4.0 Production Processes," in *IFIP Advances in Information and Communication Technology (AICT)*, Novi Sad, Serbia, 2020, vol. 591, pp. 1–8, doi: https://doi.org/10.1007/978-3-030-57993-7_50.