Integrating Modeling Languages with Ontologies in the Context of Industry 4.0

Mario Libro, Sebastiano Gaiardelli, Michele Lora, Franco Fummi
Industrial Computer Engineering Laboratory,
Department of Engineering for Innovation Medicine,
University of Verona, Italy,
name.surname@univr.it

***Abstract*—The evolving landscape of manufacturing systems and the increasing complexity of production lines necessitate innovative approaches for efficient information management and process modeling. The System Modeling Language (SysML) provides a powerful language to express such information. However, the expressiveness comes at a cost: on the one hand, the modeling phase requires a deep understanding of the domain; on the other, SysML lacks rigorous semantics.**

This work introduces a novel methodology that enriches the SysML with ontology reasoning in the context of manufacturing systems. The approach uses ontologies as a comprehensive knowledge base that encapsulates essential details about the machinery, their provided functions, and the associated constraints. The approach offers a reliable and efficient way to verify the consistency and correctness of production recipes: it ensures recipes' practical applicability in the manufacturing process while reducing errors that can occur in the modeling phase. The proposed methodology has been validated through its application to a fully-fledged manufacturing line, showing its applicability in real-world scenarios.

***Index Terms*—Computer-aided manufacturing, process modeling, knowledge representation.**

I. Introduction

Industry 4.0 is deeply changing manufacturing companies. This revolution has brought a range of unprecedented challenges for the design and operation of production lines [1]. A critical problem lies in developing effective and sophisticated information management systems [2]. Manufacturing systems are always more interconnected with each other, increasing their complexity day by day. Optimizing the flow of information between these types of systems is crucial for maintaining high production standards [3].

These challenges, along with the increasing product variants, smaller lot sizes and volatile demand require flexible production systems that can be adapted to the needs [4]. A deeper understanding of factory operations, constraints, and requirements is fundamental to assist manufacturing companies in this evolving landscape.

This study was carried out within the PNRR research activities of the consortium iNEST (Interconnected North-Est Innovation Ecosystem) funded by the European Union Next-GenerationEU (Piano Nazionale di Ripresa e Resilienza (PNRR) – Missione 4 Componente 2, Investimento 1.5 – D.D. 1058 23/06/2022, ECS_00000043). It has been also partially supported by the PRINN 2022T7YSHJ SMART-IC - Next Generation EU project. This manuscript reflects only the Authors' views and opinions, neither the European Union nor the European Commission can be considered responsible for them.

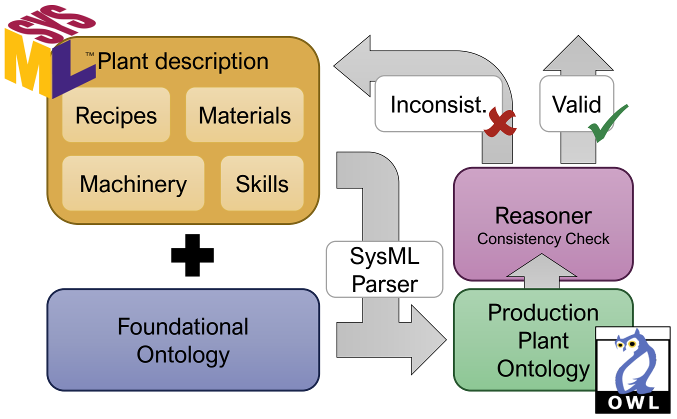

Figure 1. Overview of the proposed methodology. On the left, the production plant, represented by a SysML is paired with a Foundational Ontology to create the ontology of the production plant. A reasoner verifies the correctness of the model by checking its consistency.

Existing modeling languages, such as SysML, already provide support to represent comprehensive descriptions of machinery and manufacturing processes [5]. They provide organized and accurate ways to describe the many components of a manufacturing system, enabling better comprehension and analysis of production workflows. Nevertheless, they display an inherent limitation in their capacity to represent the semantics of factory operations with the same level of rigorousness, creating a considerable gap in the complete understanding of manufacturing processes. A promising direction lies in representing manufacturing systems with formal descriptions, able to catch their capabilities and interoperability [6], [7].

In the context of manufacturing processes, semantics refer to the detailed understanding of the transformations performed on material throughout the production process. This requires an in-depth understanding of the alterations a material undergoes as it progresses from one manufacturing stage to another, its interaction with various machines, and the subsequent outputs at each stage. Despite the rich information SysML modeling offers, it requires a significant amount of expertise. Therefore, simplifying the modeling process while preserving the integrity and accuracy of the models is a substantial need.

Figure 1 depicts the contribution of this paper. We propose a methodology combining the use of SysML and ontologies to check the consistency of production recipes. Production systems and recipes are modeled in SysML models according to the modeling approach described in [8]. The model describes the system equipment and the production recipes. The lack of formal support for SysML models is mitigated by using a *Foundational Ontology*, based on the DIN 8580 standard, defining the terminology, the concepts, and the relations among concepts. Combining the SysML model and the Foundational Ontology, the approach builds a *Production Plant Ontology* that encodes the information in the SysML models using the concepts in the Foundational Ontology. Finally, a solver is used to verify the consistency of the Production Plant Ontology. If the ontology is consistent, then the specified recipes can be executed by the production system. Otherwise, the solver provides feedback useful for correcting the recipes in the SysML model.

In Section II we analyze the state-of-the-art in information modeling and model-based design of production systems. Then, in Section III we present the structure of the Production Plant Ontology and the information it contains. Section IV presents the methodology that automatically creates the Production Plant Ontology and verifies its consistency. Section V is devoted to demonstrating the applicability of the proposed approach by modeling a production process of a fully-fledged production line. As pointed out in the concluding remarks presented in Section VI, the methodology allows to reduce the amount of work necessary to specify and verify the production recipes. Furthermore, it enables rigorous reasoning about production recipes and system capabilities.

II. Background

Model-based System Engineering (MBSE) is a methodology supporting all the design phases by using models of the system being engineered [9]. MBSE also proved useful in the manufacturing context to design complex manufacturing systems [10]. SysML [11] is the de facto standard modeling language to model complex systems implementing the MBSE principles [9], [12]. However, as a major drawback, SysML lacks formal semantics that would enable verifying system requirements, model consistency, and correctness.

Previous attempts of pairing SysML with ontologies aimed at combining the graphical and intuitive way to represent manufacturing models of SysML, with the support for consistency verification provided by ontologies [13]. SysML models are first translated into a Web Ontology Language (OWL) file; then, a set of queries check the consistency of the model [14]. Among the principal applications of this methodology, we can find requirements verification [12], [15], [16] and constraints verification [7] during all the design phases. Other works are focused on the verification of the scheduling and planning with respect to the available materials and machine capabilities [17], [18]. These works propose an ad-hoc mapping from SysML to ontologies, limiting the interoperability of the SysML models.

A promising approach, proposed in [10], [19], [20], involves building ontologies based on community-maintained Information Resources (IRs). IRs typically include industry standards such as ISO or IEEE standards, alongside scientific publications, and project reports. In particular, this approach incorporates industrial standards like ISA 88, VDI 2860, and DIN 8580. The standards, as core components of IRs, offer mature and universally acknowledged information, being widely recognized and adopted within the professional community. As such, they ensure a common terminology accepted by large groups of users familiar with these standards. Alternatively, some works that propose the use of SysML metamodels based on industrial standards to facilitate the modeling phase [21], [22].

While these works are promising, mapping from SysML to the ontology is performed manually and usually limited to requirements and constraints verification. While recent work is based on machine standards, it does not focus on machine capabilities, which is the main focus of our contribution.

III. Ontology Construction

The proposed methodology assumes SysML models based on the modeling approach described in [8]. The production system structure is modeled as a SysML Block Definition Diagram (BDD); the pieces of machinery are modeled as blocks in the BDD. Each manufacturing action performed by machines is represented by a SysML Activity Diagram. Each Activity Diagram models the transformation from the input to the output pieces. This section describes how the concepts expressed in the SysML are encoded into ontologies.

As depicted in Figure 1, the construction of a *Production Plant Ontology* requires combining the SysML description of the plant with a *Foundational Ontology*. The Foundational Ontology is a fixed ontology built by relying on the main available IRs. As such, it contains all the knowledge required to describe production plants in general. For instance, it must contain the knowledge required to express concepts such as *physical machine* and *manufacturing operations* as well as their relations. In this work, we rely on a Foundational Ontology based on the DIN 8580 standard. Then, the information modeled in the SysML description is mapped onto the classes and relations defined by the Foundational Ontology to create the Production Plant Ontology.

A class in an ontology represents a set of instances or individuals sharing common characteristics or attributes. The ontologies we propose to build are structured around four principal classes (`Machine`, `MachineFunction`, `Piece` and `PieceState`), which are defined in the Foundational Ontology. Table I summarizes the fundamental elements used to construct ontologies. Each line reports the main classes being used, grouped according to the four principal classes listed above. For each class, columns specify the subclasses inheriting from the class, their role, the relations and inverse relationships with other classes, and the range (i.e., the target class) of these relations. The class data properties column

Table I
COMPREHENSIVE REPRESENTATION OF THE ONTOLOGY STRUCTURE, DETAILING THE SCOPES, CLASSES, SUBCLASSES, OBJECT PROPERTIES AND THEIR RANGES. THE TABLE ALSO INDICATES THE PRESENCE OF INVERSE OBJECT PROPERTIES, DATA PROPERTIES, AND PROPERTY RESTRICTIONS, OFFERING A COMPLETE OVERVIEW OF THE ONTOLOGY'S ARCHITECTURE.

Concepts	**Classes**	**Subclasses**	**Object Properties**	**Range**	**Inverse Objecc Properties**	**Data Prop.**	**Prop. Restrictions**
Physical industrial machinery available in the plant	**Machine**	Taxonomy	provides-MachineFunction	Machine-Function	isProvided ByMachine	✓	✗
Manufacturing operations performed by machinery	**MachineFunction**	Taxonomy	providesOutputPiece, requiresInputPiece	Piece	isOutputPieceOf-MachineFunction, isInputPieceOf-MachineFunction	✓	✗
Materials processed by a manufacturing operation	**Piece**	InputPiece, OutputPiece	hasPieceState	PieceState	isPieceStateOf-Piece	✓	✗
Inputs materials of a manufacturing operation	InputPiece	–	hasPieceState	PieceState	isPieceStateOf-Piece	✓	✗
Materials produced by a manufacturing operation	OutputPiece	–	hasPieceState	PieceState	isPieceStateOf-Piece	✓	✗
Material's state	**PieceState**	InputPieceState, OutputPieceState	–	–	–	✓	✗
State of input materials	InputPieceState	MachineFunction-InputPieceState	–	–	–	✓	✗
Semantic of machine functions	OutputPieceState	–	–	–	–	✓	✗
Machine function's constraints	MachineFunction-InputPieceState	–	–	–	–	✓	✓

indicates whether the class has attributes, while the property restrictions column reports whether the class has constraints.

Each *physical machine* in the industrial plant, originally modeled by a SysML block, is represented in the ontology by an instance of the `Machine` class. Machines are further organized in a taxonomy represented by a hierarchy of subclasses. The right-hand side of Figure 2 depicts the class hierarchy representing a machine taxonomy. For instance, the *Milling Machine* class inherits from the *Subtractive Production Machine* class, a subclass of *Material Processing Machine* extending the class representing all the *Machines*. Each machine is related to a set of *machine function* via the `providesMachineFunction` object property and the reversed object property `isProvidedByMachine`.

The `MachineFunction` class, structured in a hierarchy of subclasses, forms a taxonomy that encapsulates the diverse operations or functions a machine can perform. Initially modeled using SysML Activity Diagrams, these manufacturing operations are integrated into the ontology by instantiating individuals within the subclass hierarchy, as illustrated on the left side of Figure 2. Machine functions are connected to machines through the relations `providesMachineFunction` and its inverse, `isProvidedByMachine`. In Figure 2, the blue boxes depict the instantiation of two individuals: one representing a machine function and the other a specific milling machine. Thus, the characteristics of each individual are enriched using data properties, allowing for the specification of a range of attributes. In the ontology, each individual representing an operation is also connected to individuals from the `Piece` class. This connection is established

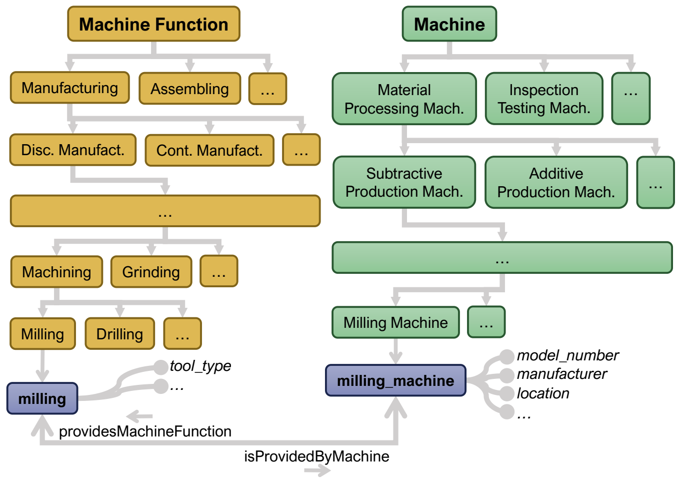

Figure 2. Portion of the Foundational ontology. The left-hand side depicts the hierarchical taxonomy of machine functions. The right-hand side shows the taxonomy of machines. Blue boxes highlight the instantiation of individuals within the *Machine* and *MachineFunction* subclasses and their connection via the `providesMachineFunction` relation.

through the object properties `providesOutputPiece` and `requiresInputPiece`.

The `Piece` class represents *materials* used or produced by machine functions. Each individual of this class is associated with individuals of the class `PieceState`, using the object property `hasPieceState`. This class is specialized by the subclasses `InputPiece` and `OutputPiece`. These subclasses represent the role of pieces as either inputs for

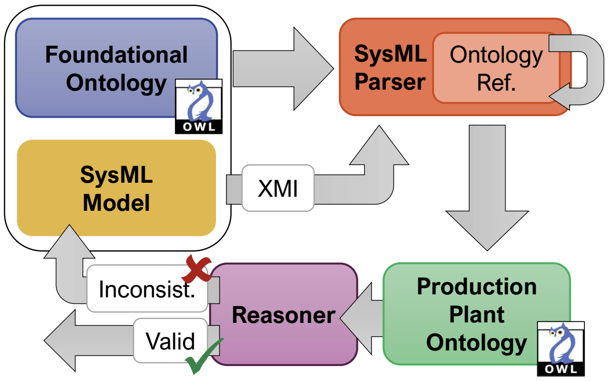

Figure 3. Overview of the verification pipeline, composed of five main activities. The Production Plant Ontology is generated by a parser analyzing the SysML model and the Foundational Ontology. A resoner verify the correctness of the Production Plant Ontology (i.e., SysML model).

machine functions or outputs from these operations.

The `PieceState` class identifies the state of a material at a specific moment. This class is concretized by the `InputPieceState` and `OutputPieceState` subclasses. The `InputPieceState` class represents the state of an input piece of a machine function. Meanwhile, the `OutputPieceState` class describes the alterations or modifications that a machine function applies to input pieces.

The `InputPieceState` class is composed of numerous subclasses (i.e., `MachineFunctionInputPieceState`), precisely one for each specific machine function instance. These subclasses allow adding restrictions on the individuals they encapsulate. This mechanism allows to define both the constraints and the effect of a machine function over its input pieces. These constraints are specified using a value restrictions technique. This ensures that the data properties of instantiated individuals within each subclass conform to particular values or ranges of values.

IV. Ontology Verification

The proposed verification pipeline is illustrated in Figure 3. A key component is the *SysML Parser*, necessary to encode SysML models into OWL ontologies. The parser takes as input a SysML model and the Foundational Ontology and generates the Production Plant Ontology. Then, it creates the Production Plant Ontology by mapping the information contained in the model onto the concepts specified in the Foundational Ontology and applying multiple ontology refinement steps. Once the SysML model is encoded in the Production Plant Ontology, its correctness is evaluated by using *Pellet* [23]: a reasoner supporting ontologies encoded in OWL 2 DL. The reasoner checks the consistency of the model to identify inconsistencies in the SysML model. Thus, ensuring that the production recipes are coherent and executable by the specified production plant. The following part of this section details the steps of the proposed verification pipeline.

Algorithm 1: Instantiate input piece states.

```
1  Function findSemantics(mf: MachineFunction):
2      return ontology.SPARQL("SELECT ?property
         ?value WHERE {mf ?property ?value .}");
3  Function instantiateIndividual(piece:
     InputPieceState, mf: MachineFunction):
4      return ontology.SPARQL(piece, mf);
   Inputs : ips_list: InputPieceStates[], mfs_list:
            MachineFunctions[], mchs_list: Machines[]
5  for i=1 to length(ips_list) do
6      if ips[i] is undefined then
7          semantics_mf ←
             findSemantics(mfs[i-1]);
8          ips[i] ← combine(semantics_mf, ips[i-1]);
9      instantiateIndividual(ips[i], mfs[i]);
10 end
```

A. SysML Parsing and Ontology Mapping

The SysML model is exported into an XML Metadata Interchange (XMI) file. The XMI format is both machine-readable and an open standard, making it an ideal choice for interoperability. The parser analyzes the XMI file to extract all the necessary information to generate the Production Plant Ontology. The creation of the Production Plant Ontology follows four steps:

1) Creation of the Foundational Ontology by establishing the static structure of the ontology. As the Foundational Ontology is based on shared IRs, this step can be done only once for multiple production plants in the case multiple production plants can be expressed by using the same set of concepts;
2) Population of the ontology. The ontology is populated with individuals by processing SysML BDD diagrams. This process consists of instantiating individuals belonging to `Machine`, `MachineFunctions`, `Pieces`, and `PieceState` classes;
3) Instantiation of the production recipes. The `MachineFunctionInputPieceState` classes are created by analyzing the SysML Activity Diagram and instantiating all the necessary individuals to represent the production recipes within the ontology;
4) Generation of the OWL file containing the Production Plant Ontology.

To construct and manipulate ontologies, the parser exploits OwlReady2 [24]: a Python library providing a set of utilities to create, analyze and manipulate ontologies. The process begins by loading the Foundational Ontology from an OWL file. Once the foundational structure is established, the parser analyzes the SysML BDDs within the XMI file. The parser translates all the machines and their operations into distinct individuals within the ontology. The individuals are organized within the specific taxonomies of the `Machine` and `MachineFunction` classes. In this phase, the constraints of each machine function are also incorporated into the ontology.

For each machine function, it creates a new subclass extending the `InputPieceState` using logical restrictions.

Then, the parser processes the Activity Diagrams of the production recipes. It creates a `MachineFunctionInputPieceState` instance for each operation specified in the recipes. The information in the SysML Activity Diagrams of the production recipes does not provide enough details of the material's state processed by the operations. Furthermore, it is worth noting that when modeling a production recipe, the user is not forced to define an input for every operation, as it often depends on the outputs of other operations. However, the complete list of inputs of each operation is crucial information to verify the consistency of the recipe. The missing pieces of information are computed by querying the intermediate ontology with a set of SPARQL queries, according to the procedure described in Algorithm 1.

The algorithm takes as input the ordered sequence of operations of a production recipe `mfs_list`, the machines executing these operations `mchs_list`, and the specified inputs for each operation `ips_list`. The algorithm searches the undefined input piece states by looping over the list containing all the input piece states (lines 5-6). An input piece state is undefined whenever not specified within the SysML Activity Diagram. Lines 7-8 compute the undefined input piece state: it first retrieves the semantics of the machine function producing the input piece state `ips[i]`; then it applies the semantics over the previous input piece state `ips[i-1]`. The function `findSemantics`, defined in line 1, retrieves the semantics of a machine function by searching it within the ontology. The semantics contain the attributes modified by the operation. In line 8, the retrieved semantics is combined with its input piece state, effectively creating the output piece state of that operation. This output state is used as the input piece state for the current operation. Then, the computed input piece state is instantiated (line 9) by calling the function `instantiateIndividual` (defined in line 3) with the current operation and input piece state as inputs. This function instantiates the individuals in their corresponding `MachineFunctionInputPieceState` class.

B. Ontology Consistency Checking

Once this step is finished, the Production Plant Ontology is completed and can be exported into an OWL file. The model's correctness and the feasibility of the production recipe are verified by checking the consistency of the ontology using the Pellet reasoner. Additionally, the OWL file can be examined using Protégé [25], a well-known tool for creating, manipulating, and analyzing ontologies. This allows us to properly examine generated ontology, enabling the analysis of the inconsistencies identified by the Pellet reasoner.

V. Case Study

The proposed methodology has been validated by modeling a production recipe of the Industrial Computer Engineering

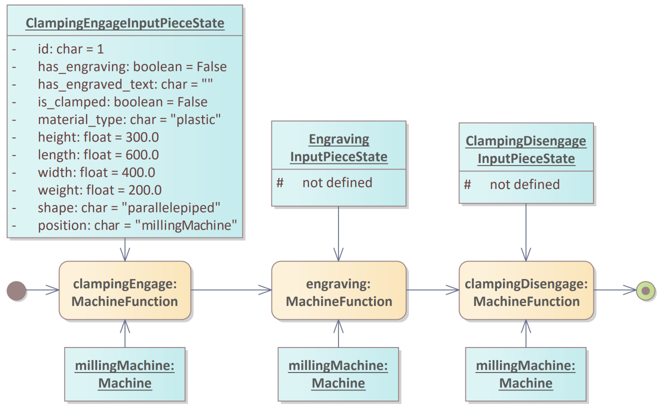
ClampingEngageInputPieceState
- id: char = 1
- has_engraving: boolean = False
- has_engraved_text: char = ""
- is_clamped: boolean = False
- material_type: char = "plastic"
- height: float = 300.0
- length: float = 600.0
- width: float = 400.0
- weight: float = 200.0
- shape: char = "parallelepiped"
- position: char = "millingMachine"
Engraving InputPieceState
# not defined
ClampingDisengage InputPieceState
# not defined
clampingEngage: MachineFunction
engraving: MachineFunction
clampingDisengage: MachineFunction
millingMachine: Machine
millingMachine: Machine
millingMachine: Machine

Figure 4. Overview of the production recipe used as case study represented with SysML activity diagram. Yellow boxes identify the machine function composing the recipe. Each machine function is connected to its accepted input piece state (on the top) and the machinery implementing the machine function (on the bottom).

(ICE) Laboratory: a research facility[1] equipped with a fully-fledged reconfigurable manufacturing line. The ICE laboratory is equipped with a *vertical automated warehouse* storing the materials; a *robotic assembly cell* with two collaborative robots; a *Quality Checking (QC) cell* with visual cameras; an *additive manufacturing cell* equipped with multiple 3D printers; a *subtractive manufacturing cell* equipped with a four-axis Computerized Numerical Control (CNC) milling machine; a *functional control cell* equipped with a flying probe functional tester machine for testing electronic boards. A mini-pallet conveyor belt and two Automated Guided Vehicles (AGVs) move materials between the different working cells in the manufacturing line.

We considered the production recipe depicted in Figure 4. It implements a manufacturing process using the milling machine to engrave a specific word onto a plastic piece. The production recipe consists of three main machine functions: `clampingEngage` operation closing the clamping device to hold the workpiece; the `engraving` operation, which engraves the specified word on the workpiece; the `clampingDisengage` operation opening the clamping device to release the workpiece.

A. Automatic Generation

Table II reports the main metrics of the SysML model used in the case study. The *initial value* column indicates the total counts of each parameter, providing a snapshot of the model before applying our methodology. The *generated* column reports the alterations in these values after applying the automatic generation process. Since the automatic generation process is applied only to the ontology, the latter column applies solely to the OWL file's parameters.

The table is organized into three sections, each representing a different set of parameter types. The first section details the parameters related to the SysML model. The second section

[1]The ICE laboratory: https://www.icelab.di.univr.it/

Table II
SUMMARY OF THE APPLICATION RESULTS.

Type	Parameter	Initial Value	Generated
SysML	# BDD	12	–
	# Activity	48	–
	# Relationships	28	–
	# ForkJoin	36	–
OWL	# XML Tags	972	1539
	# Axioms	312	462
	# Classes	57	60
	# Individuals	0	16
	# Data Properties	11	–
	# Object Properties	8	–
Generation time		284 ms	
Reasoner time (consistent ontology)		1229 ms	
Reasoner time (inconsistent ontology)		1660 ms	

outlines the parameters related to the ontologies. Finally, the last section specifies the execution time for both the automatic generation and the consistency verification phases.

The SysML model applied in our case study is intricately structured, comprising various elements essential for detailing the manufacturing machinery and processes. It includes 6 BDD, 18 relationships, 36 Fork or Join elements, and 45 activity elements.

The Foundational Ontology encapsulates a substantial amount of information. The OWL file contains 972 Extensible Markup Language (XML) tags. These tags define 325 classes, 60 individuals, and 33 types of properties. Upon applying our automated generation process to the SysML model, we observe a significant expansion in the size of the resulting Production Plant Ontology. The increased size reflects the added depth and details the proposed process introduces to the original SysML model. In our experiments, carried out on a laptop equipped with an Intel i7 10700K CPU and 32 GB RAM, the time required to generate the Production Plant Ontology is 284 ms.

Creating such an expanded ontology would be a time-consuming and error-prone task. The automatic generation process reduces the time required to manually define the ontology while also reducing the probability of mistakes. Thus, the automation of the proposed methodology reduces a substantial burden for the engineers. Furthermore, engineers can exploit the full potential of ontology reasoning without being required to have deep knowledge about ontology technologies.

B. Consistency Verification

The effectiveness of our methodology has been verified with two different experiments. First, we build a valid SysML model in which all the represented pieces of information are consistent with each other and with the Foundational Ontology. This first test demonstrates the consistency of the Foundational Ontology and that the automatic generation procedure generates consistent ontology. The time required for the Pellet reasoner to verify the consistency is 1229 ms. Then, we created a second model by introducing constraint violations within the modeled production recipe used as a case study, making the production recipe unfeasible. In this case, the Pellet reasoner correctly identified the inconsistency. The consistency check required 1660 ms, slightly more time than in the previous case due to the time necessary to provide the information about the identified inconsistencies.

```
InputPieceState
and (has_height some xsd:double
    [< "1000.0"^^xsd:double])
and (has_engraved_text value "")
and (has_engraving value false)
and (has_material_type value "plastic")
and (has_position value "millingMachine")
and (has_shape value "parallelepiped")
and (is_clamped value false)
```

Figure 5. Extract of property restrictions for the clamping operation, detailing required properties of a suitable input piece.

Consider the `ClampingEngage` machine function used in our production recipe. The constraints regarding the accepted state of the input piece are imposed using property restrictions, as shown in Figure 5. These conditions specify that the input piece must not have been already engraved, must be unclamped, must possess a parallelepiped shape, and others. In this second experiment, we create a production recipe containing an input piece violating these restrictions. Specifically, we provided in input to the `ClampingEngage` machine function a piece with the following wrong properties: `has_material_type` equals to “steel”, `has_shape` equals to “cylinder”, and `has_height` equals to “1100.0”. The reasoner recognizes and highlights the issues, describing the reasoning process used to detect the inconsistency. The results obtained from the Pellet reasoner are shown in Figure 6.

For the sake of readability, we reported only one of the three explanations for the constraint violations returned from the Pellet reasoner. The other two explanations follow a similar pattern. The displayed explanation reports the specific causes of the inconsistency. First, it identifies the individual `ClampingEngageInputPieceState` with the data property `has_shape`, which contradicts the machine function’s constraints. This leads to the conclusion that the input piece used for the `ClampingEngage` operation in the production recipe is invalid. Specifically, the value assigned to the attribute `has_shape` contains the cylinder value instead of the parallelepiped shape. This explanation guides the user towards the presence of errors within the SysML model, enabling in-advance error detection.

VI. CONCLUSIONS

In this paper, we introduced a novel methodology enriching SysML models of manufacturing systems with ontology reasoning. The proposed methodology aims to verify the correctness of information represented within SysML models by checking the consistency of an automatically generated

```
Ontology is inconsistent, run "pellet explain" to
get the reason.
This is the output of `pellet explain`:
Axiom: Thing subClassOf Nothing
Explanation(s):
1) ClampingEngageInputPieceState subClassOf
    InputPieceState
   and has_engraved_text value "()"^^string
   and has_engraving value "false"^^boolean
   and has_material_type value "plastic"^^string
   and has_position value "millingMachine"^^string
   and has_shape value "parallelepiped"^^string
   and is_clamped value "false"^^boolean
   and has_height some decimal[>= 0.0, < 1000.0]
   and has_shape exactly 1 string
   ...
   and has_engraved_text exactly 1 string

   conceptMill_clampingEngage_inputState1
    has_shape "cylinder"^^string

   conceptMill_clampingEngage_inputState1
    type ClampingEngageInputPieceState
```

Figure 6. Terminal output from Pellet reasoner, highlighting inconsistencies found in the ontology after verification.

Production Plant Ontology. We validate the proposed methodology by showing its applicability in real-world scenarios. Results showed that by exploiting our proposed methodology, it is possible to rigorously verify the feasibility of production recipes in a given system. Furthermore, the automated generation of the ontologies from the production system models decreases the effort required by engineers.

In the future, we aim to extend the Foundational Ontology by including more industrial standards. Thus, allowing the application of the methodology to more real-world scenarios.

References

[1] S. Zhanybek, S. Sabit, D. Dinara, S. Essam, and T. Ali, "Industry 4.0: Clustering of concepts and characteristics," *Cogent Engineering*, vol. 9, no. 1, p. 2034264, 2022.

[2] B. Thomas, C. John, and D. Frank, "A review of interoperability standards for industry 4.0." *Procedia Manufacturing*, vol. 38, pp. 646–653, 2019, 29th International Conference on Flexible Automation and Intelligent Manufacturing (FAIM 2019), June 24-28, 2019, Limerick, Ireland, Beyond Industry 4.0: Industrial Advances, Engineering Education and Intelligent Manufacturing.

[3] M. Wang, S. Pang, S. Yu *et al.*, "An optimal production scheme for reconfigurable cloud manufacturing service system," *IEEE Transactions on Industrial Informatics*, vol. 18, no. 12, pp. 9037–9046, 2022.

[4] E. Järvenpää, N. Siltala, and M. Lanz, "Formal resource and capability descriptions supporting rapid reconfiguration of assembly systems," in *2016 IEEE International Symposium on Assembly and Manufacturing (ISAM)*, 2016, pp. 120–125.

[5] S. Gaiardelli, S. Spellini, M. Lora, and F. Fummi, "Modeling in industry 5.0: What is there and what is missing: Special session 1: Languages for industry 5.0," in *Proc. of Forum on specification Design Languages (FDL)*, 2021, pp. 01–08.

[6] A. Köcher, C. Hildebrandt, L. M. Vieira da Silva, and A. Fay, "A formal capability and skill model for use in plug and produce scenarios," in *2020 25th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, vol. 1, 2020, pp. 1663–1670.

[7] S. Lu, A. Tazin, Y. Chen, M. M. Kokar, and J. Smith, "Detection of inconsistencies in SysML/OCL models using OWL reasoning," *SN Computer Science*, vol. 4, no. 2, Jan. 2023.

[8] S. Spellini, S. Gaiardelli, M. Lora, and F. Fummi, "Enabling component reuse in model-based system engineering of cyber-physical production systems," in *Proc. of IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, 2021, pp. 1–8.

[9] A. L. Ramos, J. V. Ferreira, and J. Barceló, "Model-based systems engineering: An emerging approach for modern systems," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 42, no. 1, pp. 101–111, 2012.

[10] C. Hildebrandt, A. Köcher, C. Küstner *et al.*, "Ontology building for cyber–physical systems: Application in the manufacturing domain," *IEEE Transactions on Automation Science and Engineering*, vol. 17, no. 3, pp. 1266–1282, July 2020.

[11] S. Friedenthal, A. Moore, and R. Steiner, *A Practical Guide to SysML, Third Edition: The Systems Modeling Language*, 3rd ed. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2014.

[12] R. Chen, C.-H. Chen, Y. Liu, and X. Ye, "Ontology-based requirement verification for complex systems," *Advanced Engineering Informatics*, vol. 46, p. 101148, 2020.

[13] X. Hu, R. Arista, X. Zheng *et al.*, "Ontology-based system to support industrial system design for aircraft assembly," *IFAC-PapersOnLine*, vol. 55, no. 2, pp. 175–180, 2022, 14th IFAC Workshop on Intelligent Manufacturing Systems IMS 2022.

[14] H. Wang, V. Thomson, and C. Tang, "Change propagation analysis for system modeling using semantic web technology," *Advanced Engineering Informatics*, vol. 35, pp. 17–29, 2018.

[15] A. Ashari, A. Sari, and H. Wardhana, "An extended rule of the sysml requirement diagram transformation into owl ontologies," *International Journal of Intelligent Engineering and Systems*, vol. 14, pp. 506–515, 02 2021.

[16] H. Wardhana, A. Ashari, and A. K. Sari, "Transformation of sysml requirement diagram into owl ontologies," *International Journal of Advanced Computer Science and Applications*, vol. 11, no. 4, 2020.

[17] M. Vegetti and G. Henning, "Ontology network to support the integration of planning and scheduling activities in batch process industries," *Journal of Industrial Information Integration*, vol. 25, p. 100254, 2022.

[18] G. Engel, T. Greiner, and S. Seifert, "Ontology-assisted engineering of cyber–physical production systems in the field of process technology," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 6, pp. 2792–2802, June 2018.

[19] C. Hildebrandt, S. Törsleff, B. Caesar, and A. Fay, "Ontology building for cyber-physical systems: A domain expert-centric approach," in *2018 IEEE 14th International Conference on Automation Science and Engineering (CASE)*, Aug 2018, pp. 1079–1086.

[20] A. Köcher, C. Hildebrandt, L. M. Vieira da Silva, and A. Fay, "A formal capability and skill model for use in plug and produce scenarios," vol. 1, pp. 1663–1670, Sep. 2020.

[21] C. Hildebrandt, A. Scholz, A. Fay *et al.*, "Semantic modeling for collaboration and cooperation of systems in the production domain," in *2017 22nd IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, 2017, pp. 1–8.

[22] Y. Feng, Q. Zou, C. Zhou, Y. Liu, and Q. Peng, "Ontology-based architecture process of system-of-systems: From capability development to operational modeling," *Applied Sciences*, vol. 13, no. 9, 2023.

[23] E. Sirin, B. Parsia, B. C. Grau, A. Kalyanpur, and Y. Katz, "Pellet: A practical owl-dl reasoner," *Journal of Web Semantics*, vol. 5, no. 2, pp. 51–53, 2007.

[24] J.-B. Lamy, "Owlready: Ontology-oriented programming in python with automatic classification and high level constructs for biomedical ontologies," *Artificial Intelligence in Medicine*, vol. 80, pp. 11–28, 2017.

[25] J. H. Gennari, M. A. Musen, R. W. Fergerson *et al.*, "The evolution of protégé: an environment for knowledge-based systems development," *International Journal of Human-computer studies*, vol. 58, no. 1, pp. 89–123, 2003.