[GWF+23] S. Greiner, B. Wiesmayr, K. Feichtinger, K. Meixner, M. Konersmann, J. Pfeiffer, M. Oberlehner, D. Schmalzing, A. Wortmann, B. Rumpe, R. Rabiser, A. Zoitl: Maturity Evaluation of Domain-Specific Language Ecosystems for Cyber-Physical Production Systems. In: 2023 IEEE 28th International Conference on Emerging Technologies and Factory Automation (ETFA), IEEE, Sep. 2023.

2023 IEEE 28th International Conference on Emerging Technologies and Factory Automation (ETFA) | 979-8-3503-3991-8/23/$31.00 ©2023 IEEE | DOI: 10.1109/ETFA54631.2023.10275624

Maturity Evaluation of Domain-Specific Language Ecosystems for Cyber-Physical Production Systems

Sandra Greiner[1], Bianca Wiesmayr[2], Kevin Feichtinger[2], Kristof Meixner[3], Marco Konersmann[4], Jérôme Pfeiffer[5], Michael Oberlehner[2], David Schmalzing[4], Andreas Wortmann[5], Bernhard Rumpe[4], Rick Rabiser[2,6], Alois Zoitl[2,6]
[1]Software Engineering Group, University of Bern, Switzerland
[2]LIT CPS Lab, Johannes Kepler University Linz, Austria
[3]CDL SQI, Institute of Information Systems Engineering, Technische Universität Wien, Austria
[4]Software Engineering, RWTH Aachen University, Germany
[5]Institute for Control Engineering of Machine Tools and Manufacturing Units (ISW), University of Stuttgart, Germany
[6]CDL VaSiCS, LIT CPS Lab, Johannes Kepler University Linz, Austria
*E-Mail: sandra.greiner@unibe.ch

***Abstract*—Engineering Cyber-Physical Production Systems (CPPSs) heavily relies on Domain-Specific Languages (DSLs), which are tailored to a specific class of problems inherent to CPPSs. DSLs enable non-programming experts to solve problems in their domain, such as modeling production processes, implementing control software, or managing the variability of production systems. A DSL ecosystem encompasses the entire infrastructure (e.g., libraries and tools) built around its language and contributes to the successful and easy adoption thereof by domain experts. We present a maturity evaluation model for DSL ecosystems serving two aims: to *developers*, it reveals missing but essential aspects of the ecosystem, and *users* (e.g., industrial companies) can evaluate the maturity of a DSL ecosystem. We propose criteria to evaluate the maturity of DSL ecosystems and apply them to existing, publicly available DSLs which have been adopted in CPPSs. The results demonstrate that all components of the model are covered and they allow for deriving hypotheses about DSL ecosystems used in CPPSs.**

***Index Terms*—Cyber-Physical Production Systems, Domain-Specific Languages, Development Tools**

I. Introduction

Cyber-Physical Production Systems (CPPSs) are modern production systems for manufacturing (custom-tailored) goods [1]. Developing CPPSs requires addressing various engineering viewpoints, including mechanical, electrical, automation engineering, and computer sciences [2]. Furthermore, the manufacturing domain increasingly enhances production facilities with various software applications, such as data acquisition from physical components, digital twins of the physical system, or AI-enabled applications. Consequently, designing and maintaining CPPSs involves engineers from diverse backgrounds, creating a multidisciplinary environment.

The financial support by the Christian Doppler Research Association, the Austrian Federal Ministry for Digital and Economic Affairs and the National Foundation for Research, Technology and Development is gratefully acknowledged. The authors of the University of Stuttgart were supported by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) [grant number 441207927]. Authors of Johannes Kepler University Linz have received funding, in parts, from the European Union's 1-SWARM project under grant agreement 871743.

Changing requirements, uncertain conditions, and evolving CPPSs [3] represent only some challenges inherent to developing and operating software for CPPSs. Domain-specific notations, specified in Domain-Specific Languages (DSLs), address the needs of each engineering viewpoint. These notations convey challenges of integrating heterogeneous artifacts created by the respective domain experts [4]. Beyond that, domain experts rather than trained software engineers typically develop the software for CPPSs. Thus, software engineering methods, such as modularization or program complexity analysis, and recent development and maintenance approaches, such as agile methods, continuous integration, and variability modeling, hardly reach the CPPS domain [4], [5].

DSLs are languages tailored to a specific domain and may not allow for creating programs beyond the scope of this domain [6]. Due to this focus, DSLs can be optimized for domain experts with little to no programming background through simplified textual or graphical syntax. Thus, DSLs are crucial in industry [6], where engineers develop processes and the required control software for production systems (Sec. II). In CPPS engineering, DSLs support the engineering process by enabling aspects, such as modeling the configurability of a production plant or developing the control software of production resources. Examples span from variability modeling languages [7] over languages based on IEC specifications [8] to architecture languages, such as MontiArc [9]. A *DSL ecosystem* comprises the entire infrastructure surrounding the DSL, such as tool environments, style guides, or its documentation.

While software engineering research established a vast knowledge on developing and designing DSLs [10] and building respective tools [6], industrial needs tend to drive an ad-hoc and uninformed development of DSL ecosystems used for CPPSs. Thus, it is crucial for DSL *developers* to assess the maturity of the DSL ecosystem to identify potentials for improvement. Moreover, for *users* with little programming background, it is challenging to identify a mature DSL ecosystem. To address both needs, this paper contributes the *maturity evaluation model* for DSL ecosystems (Sec. III). It allows

engineers from heterogeneous backgrounds determining the maturity of a DSL ecosystem for CPPSs qualitatively. To evaluate the applicability of the model, we apply it to distinct, publicly available DSL ecosystems employed in CPPS engineering (Sec. IV). We find that all components of the model are needed to evaluate maturity of a DSL ecosystem and observe qualities of ecosystems which may correlate with its maturity.

II. Background

This section illustrates various viewpoints in CPPS engineering through an example, and presents a diverse set of DSLs used in CPPS engineering. We show how to employ each DSL in the example and outline challenges arising from using *immature* DSLs ecosystems in CPPS engineering.

A. CPPS Example

To illustrate the use of DSLs in CPPS engineering, let us consider shift forks, as parts of manual transmissions, which move cuffs along pipes to the correct position so that the gears mesh for the correct gear. The *shift fork* use case [11] describes a real-world CPPS, which produces a set of shift fork types.

Engineering CPPSs requires several activities, such as suitably modeling the product portfolio or defining the control structure. For instance, the *type comparison matrix* represents a traditional artifact for product-part definitions, which compiles product types as columns and their parts (e.g., the different pipes and forks) as rows. However, such matrices usually *miss precise semantics and attribute* definitions, which may provoke misconceptions. Similarly, control software for production resources (e.g., robots welding the forks to the pipe) can be specified in languages residing at a low-level abstraction, such as Assembly. The flexibility causes programs that are hard to comprehend and debug.

These issues call for accurate models of CPPS concepts to foster easy use and understanding, to separate the concerns of the involved disciplines, and to let engineers from these disciplines collaborate over the engineering life-cycle [12]. DSLs aim to fulfill these properties and overcome the described shortcomings by addressing the defined scope of CPPS engineering, for instance, with a product definition with rich semantics. Today, CPPS engineers already use various DSLs of varying maturity, such as SysML for system conceptualization or IEC 61499 for control code (cf. Section II-B). The maturity level of these DSLs has to be suitable for an efficient and effective industrial application.

B. Examples of DSLs (Ecosystems for CPPS)

IEC 61499 [8] defines a graphical language for developing control software. The DSL helps developers to abstract the software from the physical inputs and outputs. The block-based application models consist of software components that are collected in a library and can be executed in a runtime environment that is compliant with IEC 61499. In our use case, the DSL can be employed to develop event-based control software for the robot arms, which assemble the shift fork.

SysML v1.6 [13] is a modeling language for systems engineering applications based on a subset of UML 2 [14]. SysML provides languages for specification, analysis, design, and verification as well as structural elements (denoted as blocks) that can be composed to represent the system's structure. In the shift fork use case, block diagrams can be used to model the production-resource hierarchy. Connected parts define the usage of blocks in an internal block diagram. Connectors between ports can describe the flow of items between parts. Besides a block and the internal block diagram, SysML extends UML with a parametric diagram, which can be utilized to express constraints between properties. As the standardization is not finalized yet, we do not regard SysML v2.

MontiArc [9] is a formally defined language for modeling the architecture of distributed systems through hierarchical components and connectors. A key property is its easy extensibility, allowing for integrating new functionalities with little effort. Components perform computations and are interconnected by type-directed ports. Typed components are instantiated. Similar to IEC 61499, the language can be used to define the control software of production plants. MontiArc is closely related to SpesML [15]which adds a methodology and graphical editing to the underlying concepts. In the shift fork use case, the component controlling the robot arm may get two forks as input and assemble them as an output.

Variability (Modeling) DSLs [16] describe configurable software-intensive systems. For instance, IVML [17] is a modular textual variability modeling DSL to specify the variability of service platform ecosystems based on decision modeling principles [18]. As such, varying CPPS aspects can be described and configured by selecting varying parts and a reasoner can validate these configurations. By supporting multiple product lines through shared features and composition, it allows configuring and linking artifacts with each other [19]. Complex software-intensive systems can be developed by using the modeling editors of EASy-Producer [20], the accompanying reasoning mechanism, and modeling libraries. In the shift fork use case, variability modeling DSLs can define dependencies between processes that should be executed in a given order and guide the configuration of a production line.

The **Product-Process-Resource DSL (PPR-DSL)** [21] is a language for defining three aspects of CPPSs, (i) *products* with their properties, (ii) *processes* that produce the products, and (iii) *resources* that execute the processes, often referred to as Product-Process-Resource (PPR) concept [22]. The VDI 3682 [23] is a standard to represent the PPR concept in a technology- and implementation-agnostic graphical notation for the functional view on a CPPS. It offers a basic data model for implementation. The PPR-DSL defines a concrete machine-readable representation of the PPR aspects and allows using abstract PPR concepts and specifying and evaluating constraints between the PPR aspects. In the shift fork use case, the PPR-DSL can define the products, i.e., the shift fork and the parts, such as the type of forks, the processes which assemble the parts, and the potential resources, which can be selected, such as the available robot arms.

C. Challenges with DSLs for CPPS

DSLs for CPPSs inherit the increasing complexity of the domain [4]. Engineers of the various domains involved may already use DSLs to model certain CPPS aspects. New problems may provoke the ad-hoc development of additional DSLs which have to be integrated into the engineering process, causing a significant challenge for the domain [4]. Low-quality models with improper abstraction and neglected DSL ecosystems, such as missing tool support, documentation, or tutorials, as well as discontinued maintenance of a language or its accompanying elements threaten successfully integrating and applying DSLs. Consequently, it is challenging for users to decide which DSLs to employ and to integrate into existing processes. Additional guidance is crucial to evaluate the suitability and maturity of DSLs for CPPS engineers.

III. The CPPS DSL Maturity Evaluation Model

Based on inspecting existing DSL ecosystems used in CPPS engineering, discussions at the SECPPS workshop [24], and in subsequent meetings, we developed a model for evaluating the maturity of ecosystems of DSLs for CPPSs, depicted in Fig. 1. We describe each component and enumerate quality criteria for determining its maturity degree. Components at the bottom (i.e., grammar and syntax) build the basis, which at minimum, influence the components on top of it. The components on the left (i.e., documentation, tools, and analysis) are cross-cutting concerns that may influence the components on the right (e.g., patterns, naming conventions, and style guides) and vice versa.

A. Grammar and Syntax

This component refers to the rules on constructing syntactically correct specifications in a DSL. While the abstract syntax defines the available language elements and their potential relations, the concrete syntax defines the representation of the language elements to the users. For instance, modeling languages exist with textual, graphical, projectional, or tabular concrete syntax. Metamodels [25] or grammars [26] often define the abstract syntax while grammars may define the concrete syntax of the DSL, too.

Maturity Criteria

- **GS1** Unambiguous definition of concrete syntax (except for explicit syntactic sugar)
- **GS2** Formal definition of abstract syntax
- **GS3** Support for well-formedness rules

Supporting abstract syntax and concrete syntax is essential for each DSL as languages without either are incomplete and not usable. Well-formedness rules enable refining the set of valid models of a DSL with context-specific conditions, something metamodels and context-free grammars do not support on their own. For instance, the name of a part must be unique to be identifiable in the PPR-DSL.

B. Semantics

Semantics represents the meaning of DSL elements. For instance, Petri Nets can define the semantics of UML activity diagrams. Defining a DSL's semantics requires to map its abstract syntax onto a well-understood semantic domain [27], either formally or informally (e.g., via natural language) [28].

Maturity Criteria

- **Sem1** Definition of the semantic domain
- **Sem2** Unambiguous mapping of each DSL element into the semantic domain **either**
  - denotationally, i.e., by employing mathematical constructs **or**
  - translationally, i.e., by translating them into another language (compiling), **or**
  - operationally, i.e., by assigning execution behavior.
- **Sem3** Informal definition, i.e., through natural language

Sound definition of semantics requires an unambiguous mapping of each language construct onto a given semantic domain, e.g., an algebra. Thus, *either* a denotational, translational, or operational mapping is sufficient.

Semantics need not be defined informally. Still, for understanding the mapping (e.g., a formalization), natural language descriptions may be beneficial. Thus, informal and formal definitions of semantics benefit the maturity of a DSL ecosystem.

C. Naming Conventions

Naming conventions help developers in identifying concepts and navigating unknown projects. For instance, one should be able to distinguish a part from a process at first glance. Without naming conventions individual teams may employ their own style such that comprehending another team's software might be challenging. Thus, mature DSLs allow users to identify the syntactic constructs of yet unknown programs easily.

Maturity Criteria

- **Nam1** Naming conventions documented by developers
- **Nam2** Naming conventions applied in additional material

Ideally, language developers define naming conventions (e.g., in documents added to the language specification). Besides, naming conventions may be (partially) derived from material demonstrating the usage of the language, such as tutorials.

D. Style Guides

Style guides go beyond naming conventions. They recommend how to structure programs to increase readability and comprehension of the software. For instance, listing all blocks in one row in a block diagram may cause connections which humans can hardly decipher but do not violate the syntax.

Maturity Criteria

- **Sty1** Specification of code style documented by developers
- **Sty2** Application of coding styles in additional material

As with naming conventions, developers may explicitly document or additional material (from third parties) may demonstrate best practices of formatting programs in the DSL.

E. Libraries

Libraries allow for integrating third-party implemented functionality of recurring use. In DSLs for CPPS, problems recur, for instance, in the IEC 61499, types recur which system libraries provide as function blocks. In a mature DSL

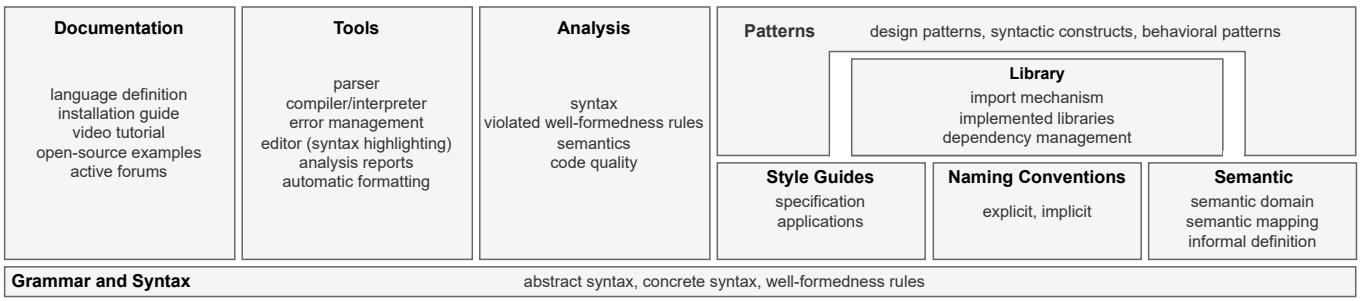

Fig. 1. Maturity evaluation model for DSL ecosystems for CPPSs.

ecosystem, developers, tool vendors, or active communities offer libraries that can be easily integrated into a program.

Maturity Criteria

- **Lib1** Syntactic construct for *importing* library functions
- **Lib2** Library implementations in the DSL **(i)** exist, **(ii)** are available, and **(iii)** actively maintained or up-to-date
- **Lib3** Dependency management between libraries

A mature DSL offers existing and available libraries, such as a standard library. Libraries have to be up-to-date when being adopted, either due to active maintenance or high stability. Additionally, an import mechanism and dependency management are indispensable to integrate (complex) libraries.

F. Patterns

(Software) design patterns [29] provide a reusable solution and may encode best practices to recurring problems. While architectural patterns structure the implemented system in a modular way, behavioral patterns allow for easily extending the behavior at runtime. For instance, the same set of block arrangements may serve to define the control unit for a robot arm. Patterns surround the component libraries because they may be employed by libraries but can also use functionality implemented in a library (e.g., standard types).

Maturity Criteria

- **Pat1** Documented design patterns
- **Pat2** Syntactic constructs to implement patterns
- **Pat3** Documented implementation patterns

As software design patterns [29] may not be appropriate in specific domains, we do not ask for their adoption for DSLs. Yet, recurring problems requiring recurring solutions are inherent to the CPPS domain. Thus, a mature DSL ecosystem should document design patterns. Additionally, syntactic constructs for implementing them are essential. Finally, implementation patterns exceed the architectural and behavioral level by defining implementation best practices.

G. Analyses

This component refers to quantitatively evaluating how a program in the DSL performs in various aspects. Analyses may identify software parts that require maintenance and lay the foundation for automated design decisions (e.g., suggesting refactorings). For instance, in textual languages, counting the lines of code gives a rough estimate of the complexity of a program. Further, analyses may examine naming conventions, style guides, or patterns. A mature DSL ecosystems documents metrics for analysis (implemented in accompanying tools).

Maturity Criteria require analyses (e.g., via **metrics**) for:

- **A1** Syntax (e.g., length (LOC) and redundancy)
- **A2** Violations of well-formedness rules
- **A3** Semantics
- **A4** Code quality, which measures the adherence to (**i**) naming conventions, (**ii**) style guides, the employment of (**iii**) design patterns, and (**iv**) libraries.

To analyze a program in the DSL, maturity criteria require *metrics* for syntax and semantics at minimum. For syntax, the total number of elements/lines of code may be measured as well as the redundancy of implementation blocks, which could be refactored into a single block. Similarly, violations of well-formedness rules can be validated and reported. For instance, analyzing the semantics of a component-based language may indicate flooding of a port through an increasing number of incoming messages. Besides, if libraries, patterns, naming conventions, and style guides exist, a program may also be analyzed for employing respective mechanisms.

H. Tools

Each component of the maturity evaluation model requires different tool support aspects. The component tools is related to documentation as users of tools require guidance for installing and using tools for developing programs in the DSL.

Maturity Criteria

- **T1** Parser for complete syntax definition
- **T2** Compiler or interpreter
- **T3** Error management and quick fixes
- **T4** Editor with at least syntax highlighting
- **T5** Analyses and reporting of the supported metrics (cf. **(i) A1** - **(iv) A4**)
- **T6** Automatic formatting according to (**i**) style guides, (**ii**) naming conventions

Developers benefit most from tools which offer the entire spectrum of support to increase their productivity.

I. Documentation

This component represents a cross-cutting concern, which benefits developing programs in the DSL. At minimum, the current language specification should be available. Ideally, further material, such as tutorials or screencasts, is available for

easily adopting the DSL. Accompanying documentation may specify naming conventions and style guides. Developers or active user communities may contribute further documentation artifacts, such as forums or tutorials.

Maturity Criteria

- **D1** Publicly available standard or language definition potentially including further documents on naming conventions and style guides
- **D2** Installation guide, which is **(i)** up-to-date and **(ii)** platform-independent
- **D3** Video tutorials or screencasts
- **D4** Open-source example projects including source code
- **D5** Active forums for exchange with users or developers

Documentation must be accessible to users to ease learning the DSL. Ideally, it is publicly available for users to familiarize with the language before selecting it for a specific use case. Available examples and active user forums help to introduce the language into new domains and use cases.

IV. Application of the Maturity Evaluation Model

This section describes how to employ the maturity evaluation model to determine the maturity of DSLs ecosystems for CPPS. It explains how to compute a metric for each of the model's components and demonstrates the results of applying the evaluation to the DSLs introduced in Sec. II-B.

A. Methodology

To evaluate the maturity of a DSL ecosystem, we consider each component of the maturity evaluation model and the respective quality criteria. For instance, the component *Patterns* requires documented design and implementation patterns as well as syntactic constructs to implement them. For computing the value of a component, the DSL ecosystem can $score$ 0, 0.5, or 1 point in each maturity criterion. For instance, if the ecosystem satisfies maturity with respect to the component *semantics*, the language will define a semantic domain, the mapping between language construct and semantic domain element, and an informal description of the mapping. Furthermore, *and-* and *or-groups* exist as sub-criteria for scoring. For instance, the mapping of semantics can be realized differently, such that it suffices to satisfy one of the sub-criteria of the mapping criterion (or-group). In contrast, in and-groups, each of the sub-criteria should be satisfied by a mature ecosystem (e.g., metrics for code quality (**A4**)).

We categorize the maturity of each component in three levels: immature, intermediately mature, and highly mature. To compute the maturity level, we divide the computed score into quartils, meaning $score < 0.25$ (○) implies immature, $0.25 >= score < 0.75$ (◓) implies intermediately mature, and $score >= 0.75$ (●) implies highly mature.

For each DSL, one author who has either worked with the language for at least 3 years or contacted one of the developers of the language, assigned the scores. A second author checked the scoring for completeness and correctness.

TABLE I
Comparison of DSLs according to the maturity model

	GS	Sem	Nam	Sty	Lib	Pat	A	T	D
IEC 61499	◓	●	◓	○	◓	●	◓	◓	◓
SysML v1.6	◓	◓	◓	◓	●	◓	○	◓	●
MontiArc	●	●	●	◓	●	○	◓	◓	●
IVML	◓	●	○	○	◓	◓	◓	◓	◓
PPRDSL	◓	●	◓	○	◓	○	◓	◓	◓

B. Application of Model to CPPS DSL Ecosystems

To examine the applicability of the maturity evaluation model, we applied it to the DSL ecosystems presented in Sec. II-B. Tab. I summarizes the results. We provide the detailed scoring online [30] and explain important decisions here.

IEC 61499. The standard defines the grammar and syntax of IEC 61499 whereas the graphical syntax is only provided as recommendations [31] [32], such that the implemented semantics and graphical syntax vary among tools. A modular library concept is not standardized but proposed in a publication [33] and two open-source tools (Eclipse 4diac[1] and FBDK[2]). In contrast to naming conventions or style guides, patterns are published in the literature (e.g., [34]). Analysis metrics are mainly described theoretically. A broad community employs IEC 61499 and various vendors offer mature tools including feasible documentation.

SysML v1.6 partially satisfies syntax and grammar by providing an abstract and concrete syntax for its syntactic elements missing well-formedness rules. SysML does not define semantics formally, but informally defines the intention of syntactic constructs. Few naming conventions exist regarding the meta-data of SysML models, for instance, the title of a model should convey which kind of model is used. Thus, this criterion is only partially satisfied (score: 0.5). Style guides and patterns are also only partially available. SysML offers a comprehensive library of models for different use cases, thereby, satisfying the respective criterion. Metrics for analysis are not provided whereas several tools (e.g., MagicDraw, or Sparx Enterprise Architect) exist for creating SysML models. As a large community employs SysML, a plethora of documentation, such as video tutorials and forums, is available.

MontiArc defines its concrete and abstract syntax using MontiCore [35] grammars, which are used to derive the parser, symbol table, and language infrastructure. Handwritten context conditions ensure a model's well-formedness in context, including a strong type system. Its semantic domain is formally defined through the FOCUS calculus [36], which is the theory of stream processing functions. MontiArc employs automata and their composition as behaviour descriptions, for which both denotational and operational semantics exist [37]. It also features a code generator that translates component models into Java simulations [38]. Language conventions are defined [38] and implemented to provide explicit feedback.

[1]Eclipse 4diac, https://www.eclipse.org/4diac/
[2]Function Block Development Kit, https://holobloc.com/fbdk11/index.htm

Style guides are not explicitly stated but defined implicitly through a pretty printer and mentioned and applied in the documentation's examples. MontiArc documents some libraries, but not all of them are up-to-date. Library management is provided through a Gradle plugin. Analyses are implemented for well-formedness rules via context conditions, and additional analyses can be implemented using a prepared framework. Documentation is provided online and as a book. [38].

IVML und Variability (Modelling) DSLs. The full specification of the textual variability modeling language IVML [17], including the language syntax and semantics, represents the only documented resource and available online[3]. Naming conventions, style guides, patterns, and libraries are not documented. The modular product line engineering tool EASy-Producer [20] provides various editors and reasoners for software ecosystems modeled with IVML. Thus, our maturity model classifies IVML as mature with respect to grammar and syntax, and semantics, but as immature in terms of style guides, naming conventions, patterns, and libraries. The tools provided for IVML are currently limited to parsers and simple reasoners in the context of EASy-Producer, but without checking well-formedness or advanced analysis capabilities. Hence, the model classifies tools and analyses as partly supported.

Product-Process-Rescource DSL. The PPR-DSL is mature in syntax and grammar but misses well-formedness rules. It satisfies semantics as it builds on and extends the VDI 3682 standard and additional constructs map onto the semantics of the variability modeling domain, such as implications or exclusions. The PPR-DSL offers little support for naming conventions and style guides as the modeled concepts possess concise nouns per default. The PPR-DSL provides basic support for libraries with an import statement and rudimentary libraries for distinct use cases. The PPR-DSL does not support patterns or metrics for analysis, but through an open-source implementation of the model, a parser, and some support libraries for auto-completion in a text editor, it provides basic tool support (0.29) including documentation.

V. Discussion

This section discusses applying the maturity evaluation model and states observations and threats to validity.

A. Observations

By applying the maturity evaluation model, we observed the following aspects: First, at least one DSL ecosystem satisfies each component. While style guides, naming conventions and libraries are rarely available in the examined ecosystems for CPPS, at least one DSL scores each component with a minimum of 0.5 points. Thus, we draw the conclusion that the components of our maturity model are chosen well.

Second, SysML and MontiArc satisfy the most components, rendering them mature DSL ecosystems for CPPS overall. In both ecosystems, either a large group of academics is involved in the development or the adoption in industry formed a large community to drive the development. The open availability of both, language specification and tools, eases contributing to the ecosystem development. The opposite can be observed for the PPR-DSL which is a comparably young DSL with few developers. Thus, the number of developers, its open accessibility, and an active community which adopts the DSL may influence the maturity degree of the DSL ecosystem. On top, a language design which allows for easy extensibility, such as MontiArc, may benefit its progress and adaption.

Third, as the leveled structure of the maturity evaluation model indicates, some components influence each other. For instance, implementation patterns may be considered as part of a style guide but still represent a pattern how to define a program such that we added them to the component patterns. Similarly, patterns may be employed to implement libraries while library functionality may be used to define a pattern (e.g., using types of a standard library). The same holds for analyses: While the component enforces metrics to analyze a language instance, without tool support, which performs the analyses automatically, the metrics remain a mere theoretical construct requiring handcrafting for their adoption. Thus, to apply the model adequately, one has to understand the contents and semantics of its components and their relationships.

Finally, language ecosystem maturity evaluation must be based on the modeling purpose (see [39]). We apply the model to languages with different purposes. For instance, the PPR-DSL names language constructs concisely such that additional naming conventions may not be necessary to distinguish a process from a part. Thus, it remains the developers' and users' task to regard the result of maturity in certain components.

B. Threats to Validity

We identified internal and external threats to validity [40] of applying the model and regarding the maturity model itself.

Internal threats of the maturity evaluation model. The evaluation criteria resulted from discussions among the authors of the paper. Our diverse academic CPPS and software engineering backgrounds and experiences with using DSL ecosystems for CPPS engineering in practice influenced the result. Claiming the criteria complete or representative requires subsequent studies (e.g., applying them to more DSLs or interviews with domain experts).

Similarly, dependencies exist between the criteria of a component which may influence the scoring. For instance, without defining a semantic domain, the mapping onto language constructs is not possible, such that the mapping cannot be satisfied if the domain is missing the component semantics. By introducing and- and or-groups for sub-criteria, we tried to counter that threat to account for fair maturity judgment. Still, we argue that it must be straightforward to check the criteria. Introducing or explicating additional dependencies may hinder rather than benefit the model's conciseness.

Internal threats of *applying* the maturity evaluation model. The authors who applied the maturity model to a language were mostly developers of the DSL. While they are the most experienced, they may be biased in turning the

[3]EASy-Producer – https://github.com/SSEHUB/EASyProducer

results towards higher maturity. Additionally, errors may have been introduced by chance or due to missing knowledge. To counter both threats, we asked the DSL developers, considered publicly available sources, when possible, and at least a second author validated the scoring independently. So far, we have not validated their agreement [41] but plan to perform an extended evaluation of the applicability with (non-academic) users of DSLs ecosystems for CPPS.

External threats regard the generalizability of the maturity evaluation model. First, the criteria of each component may be misinterpreted by third parties. To counter this threat, a subgroup of the authors discussed and developed the criteria and let the remaining authors apply the criteria to at least one DSL. In this way, we refined and clarified the criteria for scoring, for instance, by introducing different scoring for sub-criteria. As DSL users with little or no programming background have not used the maturity model yet, we plan to examine the applicability of the model with practitioners as next step.

Second, while we applied the model to several DSL ecosystems for CPPS, some of the evaluation criteria may not fit or miss an important aspect. We countered this threat by applying the model to a broad set of DSLs and refining it based on initial results and discussions. In future work, we plan to apply the model to a broader set of DSLs which may include DSLs outside the CPPS domain for comparison purposes.

VI. Related Work

To the best of our knowledge, no work evaluates the maturity of DSL ecosystems. Instead, guidelines for developing DSLs, maturity models for organizational processes and comparison criteria in software language engineering exist.

Guidelines for DSL Specification. A systematic mapping study on best practices in domain-specific modeling [42] reports guidelines and best practices that cover all phases of developing DSLs. However, it presents properties of DSLs definitions and best practices during developing DSLs only.

Another work [43] describes quality characteristics of DSLs and an assessment method to check them based on which a success level of the DSL is derived. Unlike our maturity model, the ecosystem (e.g., tools or patterns) is not considered.

Guidelines for DSL Development Processes. An approach to systematically develop DSLs [44] reports on the language definition steps, such as specifying concrete and abstract syntax and giving models meaning by defining their behavior. Beyond the language definition in form of syntax and semantics, no further guidelines are provided.

Mernik et al. [45] examine patterns for the DSL development stages: decision, analysis, design, and implementation. They discuss tools that support developers at these stages and state challenges of reusing existing General Purpose Languages (GPLs) and DSLs for creating new ones, and reducing the efforts to learn metalanguages for the DSL creation.

A survey among DSL developers [46] identifies established practices in the life-cycle of DSLs. Accordingly, comprehensive documentation, training material, and an attractive development environment encourage users to employ a DSL. Yet, the authors do not propose a method or metrics to enable future DSL developers measuring or improving these practices.

In addition to general guidelines for DSL development processes, specific guidelines and patterns for single phases of the process exist, such as monitoring [47], or theory for understanding the domain covered by the DSL [48] in the design phase, or to measure the maturity of grammars [49].

Applying DSL Guidelines. One work applies DSL guidelines and software quality attributes to assess the current state of the language design of SysML v2 [50]. It maps existing DSL guidelines onto categories of the ISO 25010 which classifies software quality attributes. The results suggest to improve the extensibility of SysML v2 in the investigated state through modularization and to provide a semantic mapping to understand the models' meaning.

Language Workbenches generate the infrastructure required to employ programming languages. Existing frameworks [51] and studies [52] offer criteria to compare such workbenches. While we do not aim for assessing workbenches, we may enrich our model with suitable criteria thereof.

Maturity Models [53], [54], such as the TRL [55] or CMM [56] are layered models to evaluate the level of maturity of organizational processes or tools. Contrary to our model, they do not consider ecosystems but rather the *evolution* of one aspect of an organization or software.

VII. Conclusion

This paper contributes the maturity evaluation model for DSL ecosystems for CPPS. As mainly non-programming experts from diverse backgrounds employ DSLs for CPPSs and due to a partial ad-hoc development of these DSL to satisfy specific uprising needs, many DSLs are unstable and ambiguous in their definition. To assess the maturity of the surrounding ecosystems, we presented the proposed evaluation model. We successfully applied the model to several DSL ecosystems, which may be employed in CPPSs. The results show that all proposed categories are applicable and the relations are well-defined. Furthermore, we observed that large-scale adoption benefits the maturity of DSL ecosystems (e.g., through advanced tools). The model resulted from discussions among academics and was qualitatively applied to a small set of publicly available DSL ecosystems. Future work shall conducts studies with practitioners from industry and apply the model in a more fine-grained way.

References

[1] S. Biffl, D. Gerhard, and A. Lüder, "Introduction to the Multi-Disciplinary Engineering for Cyber-Physical Production Systems," in *Multi-Disciplinary Engineering for Cyber-Physical Production Systems*. Springer, 2017.

[2] X. Wu, V. Goepp, and A. Siadat, "Concept and engineering development of cyber physical production systems: a systematic literature review," *The Int. Journal of Advanced Manufacturing Technology*, 2020.

[3] T. Bureš, D. Weyns, B. Schmer *et al.*, "Software engineering for smart cyber-physical systems: Challenges and promising solutions," *SIGSOFT Softw. Eng. Notes*, vol. 42, no. 2, jun 2017.

[4] K. Feichtinger, K. Meixner, F. Rinker *et al.*, "Industry voices on software engineering challenges in cyber-physical production systems engineering," in *Int. Conf. on Emerging Technologies and Factory Automation, ETFA 2022*. IEEE, 2022.

[5] T. Berger, J.-P. Steghöfer, T. Ziadi *et al.*, "The state of adoption and the challenges of systematic variability management in industry," *Empirical Software Engineering*, vol. 25, no. 3, 2020.

[6] M. Fowler, *Domain-specific languages*. Pearson Education, 2010.

[7] M. Raatikainen, J. Tiihonen, and T. Männistö, "Software product lines and variability modeling: A tertiary study," *Journal of Systems and Software*, vol. 149, 2019.

[8] Int. Electrotechnical Commission, "IEC 61499-1, Function Blocks - part 1: Architecture: Edition 2.0," Geneva, 2012.

[9] A. Butting, A. Haber, L. Hermerschmidt *et al.*, "Systematic Language Extension Mechanisms for the MontiArc Architecture Description Language," in *European Conf. on Modelling Foundations and Applications. ECMFA 2017*. Springer, 2017.

[10] A. Wasowski and T. Berger, *Domain-Specific Languages: Effective Modeling, Automation, and Reuse*. Springer Int. Publishing, 2023.

[11] K. Meixner, K. Feichtinger, R. Rabiser, and S. Biffl, "A reusable set of real-world product line case studies for comparing variability models in research and practice," in *Int. Systems and Software Product Line Conf.. SPLC 2021*. ACM, 2021.

[12] L. Berardinelli, A. Mazak, O. Alt *et al.*, "Model-driven systems engineering: Principles and application in the CPPS domain," in *Multi-Disciplinary Engineering for Cyber-Physical Production Systems*. Springer, 2017.

[13] Object Management Group, "Systems Modeling Language (OMG SysML): Version 1.6," November 2019.

[14] Object Management Group (OMG), *Unified Modeling Language. Version 2.5.1*, formal/2017-12-05 ed., Needham, MA, 12 2017, https://www.omg.org/spec/UML/2.5.1/PDF.

[15] TUM, "Spesml," 2023. [Online]. Available: https://spesml.github.io/

[16] M. H. t. Beek, K. Schmid, and H. Eichelberger, "Textual Variability Modeling Languages: An Overview and Considerations," in *Int. Systems and Software Product Line Conf.. SPLC 2019*. ACM, 2019.

[17] H. Eichelberger and K. Schmid, "IVML: A DSL for Configuration in Variability-Rich Software Ecosystems," in *Int. Conf. on Software Product Line. SPLC 2015*. ACM, 2015.

[18] K. Schmid, R. Rabiser, and P. Grünbacher, "A comparison of decision modeling approaches in product lines," in *Int. Workshop on Variability Modelling of Software-Intensive Systems*. ACM, 2011.

[19] K. Meixner, K. Feichtinger, R. Rabiser, and S. Biffl, "Efficient Production Process Variability Exploration," in *Int. Working Conf. on Variability Modelling of Software-Intensive Systems. VaMoS 2022*. ACM, 2022.

[20] H. Eichelberger, S. El-Sharkawy, C. Kröher, and K. Schmid, "EASy-Producer: Product Line Development for Variant-Rich Ecosystems," in *Int. Software Product Line Conf.: Companion Volume for Workshops, Demonstrations and Tools. SPLC 2014*. ACM, 2014.

[21] K. Meixner, F. Rinker, H. Marcher *et al.*, "A Domain-Specific Language for Product-Process-Resource Modeling," in *Int. Conf. on Emerging Technologies and Factory Automation. ETFA 2021*. IEEE, 2021.

[22] M. Schleipen, A. Lüder, O. Sauer *et al.*, "Requirements and concept for plug-and-work," *at-Automatisierungstechnik*, vol. 63, no. 10, 2015.

[23] "VDI/VDE 3682: Formalised process descriptions." Beuth Verlag, VDI/VDE, 2005.

[24] R. Rabiser, B. Vogel-Heuser, M. Wimmer *et al.*, "Workshop on Software Engineering in Cyber-Physical Production Systems (SECPPS), 2nd Edition," in *Software Engineering*, ser. LNI, vol. P-320. Gesellschaft für Informatik e.V., 2022.

[25] T. Kühne, "Matters of (meta-)modeling," *Software and Systems Modeling*, vol. 5, no. 4, 2006.

[26] K. Hölldobler, B. Rumpe, and A. Wortmann, "Software language engineering in the large: towards composing and deriving languages," *Computer Languages, Systems & Structures*, vol. 54, 2018.

[27] B. Combemale, R. France, J.-M. Jézéquel *et al.*, *Engineering Modeling Languages: Turning Domain Knowledge into Tools*. Chapman & Hall/CRC Innovations in Software Engineering and Software Development Series, November 2016.

[28] D. Harel and B. Rumpe, "Meaningful modeling: What's the semantics of "semantics"?" *Computer*, vol. 37, no. 10, 2004.

[29] E. Gamma, R. Johnson, R. Helm *et al.*, *Design patterns: elements of reusable object-oriented software*. Pearson Deutschland GmbH, 1995.

[30] S. Greiner, B. Wiesmayr, K. Feichtinger *et al.*, "Dataset: Maturity Evaluation of Domain-Specific Language Ecosystems for Cyber-Physical Production Systems," 2023.

[31] *IEC 61499-1: Function blocks - Part 1: Architecture.* International Electrotechnical Commission, 2012. [Online]. Available: https://webstore.iec.ch/publication/5506

[32] *IEC 61499-2: Function blocks - Part 2: Software tool requirements*. International Electrotechnical Commission, 2012. [Online]. Available: https://webstore.iec.ch/publication/5507

[33] M. Oberlehner, V. Ashiwal, A. Zoitl, and J. H. Christensen, "Using modules to manage the content of iec 61499 type libraries," in *Int. Conf. on Industrial Informatics. INDIN 2022*. IEEE, 2022.

[34] L. Sonnleithner, B. Wiesmayr, V. Ashiwal, and A. Zoitl, "Iec 61499 distributed design patterns," in *2021 26th IEEE Int. Conf. on Emerging Technologies and Factory Automation (ETFA)*. IEEE, 2021.

[35] K. Hölldobler, O. Kautz, and B. Rumpe, *MontiCore Language Workbench and Library Handbook: Edition 2021*, ser. Aachener Informatik-Berichte, Software Engineering, Band 48. Shaker Verlag, May 2021.

[36] M. Broy, *Specification and Development of Interactive Systems Focus on Streams, Interfaces, and Refinement*. Springer New York, 2001.

[37] B. Rumpe, "Formale Methodik des Entwurfs verteilter objektorientierter Systeme," in *Ausgezeichnete Informatikdissertationen*. B. G. Teubner Stuttgart, 1997.

[38] A. Haber, *MontiArc - Architectural Modeling and Simulation of Interactive Distributed Systems*, ser. Aachener Informatik-Berichte, Software Engineering, Band 24. Shaker Verlag, September 2016.

[39] M. Broy and B. Rumpe, "Development use cases for semantics-driven modeling languages," *Communications of the ACM*, vol. 66, no. 5, p. 62–71, 2023.

[40] C. Wohlin, P. Runeson, M. Höst *et al.*, *Experimentation in Software Engineering*. Springer, 2012.

[41] J. R. Landis and G. G. Koch, "The measurement of observer agreement for categorical data," *biometrics*, pp. 159–174, 1977.

[42] G. Czech, M. Moser, and J. Pichler, "Best practices for domain-specific modeling. a systematic mapping study," in *Conf. on Software Engineering and Advanced Applications. SEAA 2018*. IEEE, 2018.

[43] G. Kahraman and S. Bilgen, "A framework for qualitative assessment of domain-specific languages," *Software & Systems Modeling*, 2015.

[44] M. Strembeck and U. Zdun, "An approach for the systematic development of domain-specific languages," *Software: Practice and Experience*, vol. 39, no. 15, 2009.

[45] M. Mernik, J. Heering, and A. M. Sloane, "When and how to develop domain-specific languages," *ACM computing surveys (CSUR)*, 2005.

[46] H. S. Borum and C. Seidl, "Survey of established practices in the life cycle of domain-specific languages," in *Int. Conf. on Model Driven Engineering Languages and Systems*, 2022.

[47] Z. Drey and C. Teodorov, "Object-oriented design pattern for dsl program monitoring," in *Int. Conf. on Software Language Engineering (SLE)*, 2016.

[48] S. De Kinderen, "Using Grounded Theory for Domain Specific Modelling Language Design: Lessons Learned from the Smart Grid Domain," in *The Practice of Enterprise Modeling: IFIP WG 8.1. Working Conf.. PoEM*. Springer, 2017.

[49] V. Zaytsev, "Grammar maturity model." in *ME@ MoDELS*, 2014.

[50] N. Jansen, J. Pfeiffer, B. Rumpe *et al.*, "The Language of SysML v2 under the Magnifying Glass," *Journal of Object Technology*, 2022, european Conf. on Modelling Foundations and Applications. ECMFA.

[51] S. Erdweg, T. van der Storm, M. Völter *et al.*, "Evaluating and comparing language workbenches: Existing results and benchmarks for the future," *Comput. Lang. Syst. Struct.*, vol. 44, pp. 24–47, 2015.

[52] A. Iung, J. Carbonell, L. Marchezan *et al.*, "Systematic mapping study on domain-specific language development tools," *Empir. Softw. Eng.*, vol. 25, no. 5, pp. 4205–4249, 2020.

[53] E. Rios, T. Bozheva, A. Bediaga, and N. Guilloreau, "Mdd maturity model: A roadmap for introducing model-driven development," in *Model Driven Architecture – Foundations and Applications*, A. Rensink and J. Warmer, Eds. Springer, 2006, pp. 78–89.

[54] F. Tomassetti, M. Torchiano, A. Tiso *et al.*, "Maturity of software modelling and model driven engineering: A survey in the italian industry," in *16th Int. Conf. on Evaluation & Assessment in Software Engineering, EASE 2012*. IEEE, 2012, pp. 91–100.

[55] J. C. Mankins *et al.*, "Technology readiness levels," *White Paper, April*, vol. 6, 1995.

[56] K. D. Shere and M. J. Versel, "Extension of the SEI software capability maturity model to systems," in *18th Annual Int. Computer Software and Applications Conference, COMPSAC 1994*. IEEE, 1994, pp. 195–200.