Modeling Modular System Architectures with SysML v2: Towards Configuration-Ready System Models

Matthias Hüls[1,*], Luca Häfner[2], Ibrahim Parvez Thamimul Ansari[3], Hoc-Khiem Trieu[3], Eckhard Kirchner[2], Dieter Krause[1]

[1] Institute of Product Development and Mechanical Engineering Design, Hamburg University of Technology, Hamburg, Germany
[2] Institute of Product Development and Machine Elements, Technical University of Darmstadt, Darmstadt, Germany
[3] Institute of Microsystems Technology, Hamburg University of Technology, Hamburg, Germany

** Corresponding author:*
Matthias Hüls
Hamburg University of Technology
Institute of Product Development and Mechanical Engineering Design
Denickestraße 17
21073 Hamburg, Germany
☎ *040/306012176*
✉ *matthias.huels@tuhh.de*

Abstract

Modular system architectures support reuse and variant formation, but do not automatically provide the information required for computational configuration. This paper defines configuration readiness as the explicit, consistent, and machine-accessible representation of configuration-relevant information. Six requirements are synthesized from the literature and mapped to a four-level SysML v2 approach separating domain foundations, modular architecture, technical variants, and project-specific configurations and individuals. A sensor-integrating bolt illustrates the approach through manual inspection of selection and extension scenarios. The example provides a conceptual feasibility assessment.

Keywords

SysML v2, modular system architecture, model-based systems engineering, system configuration, variability modeling

 | DOI: 10.35199/dfx2026.28

1. Introduction and Motivation

Technical systems are increasingly offered in multiple variants to address heterogeneous customer, application, and lifecycle requirements. Modular product and system architectures provide an established means of managing this variety by structuring product families into reusable elements and defined combinations [1, 2]. Beyond system decomposition, this requires the explicit representation of architecture roles, available alternatives, admissible combinations, and their relation to specific system configurations.

In digital engineering, these relationships should be represented consistently in system models rather than being distributed across separate documents, tool-specific data structures, or implicit engineering knowledge. Model-Based Systems Engineering supports the integrated representation of system structures, properties, interfaces, requirements, and their relationships [3]. Product Line Engineering and Model-Based Product Line Engineering complement this perspective through systematic reuse and variability management [4, 5], while previous studies connect modular system models with configuration-related information [6, 7].

Nevertheless, a modular system model does not necessarily provide a sufficient basis for computational configuration. It may represent the system structure without explicitly identifying configuration decisions, admissible alternatives, or constraints. Reusable definitions, architecture-specific usages, selectable variants, and project-specific configurations may therefore remain ambiguous or distributed across separate representations, requiring configuration tools to reconstruct implicit relationships.

This paper investigates modular product-family architectures in SysML v2 as an information basis for computational configuration. SysML v2 provides concepts such as definitions and usages, specialization, variability, properties, and constraints [8]. However, their availability alone does not determine which configuration-relevant information must be represented or how elements with different semantic responsibilities should be organized and related.

The objective is to derive corresponding modeling requirements and operationalize them through a consistent SysML v2 modeling structure. Configuration readiness denotes the explicit, internally consistent, and machine-interpretable representation of the elements, relationships, and constraints required by a subsequent configuration process; it does not imply that the model itself generates or optimizes configurations. The contribution lies in the requirement-based organization of existing SysML v2 concepts for configuration-ready modular system models.

2. Background and Related Work

Product architecture describes the arrangement of functional elements, their mapping to physical components, and the interfaces between interacting components [9]. Modular product architectures use these principles to manage external product variety while limiting internal variety [1, 2]. Product families therefore rely on reusable elements, defined interfaces, and admissible combinations. However, a modular architecture does not automatically constitute a computational configuration model. It additionally requires explicit alternatives, configuration decisions, and constraints on the solution space.

Product Line Engineering distinguishes common and variable assets and separates reusable domain knowledge from the derivation of individual products [4]. Model-Based Product Line Engineering connects variability with engineering models [5]. Knowledge-based configuration represents selectable elements, their properties and relationships, and rules restricting admissible combinations [10, 11].

Separate representations can increase maintenance effort and cause inconsistencies when architectures, properties, or constraints change. Shafiee et al. generate product-model documentation from implemented configuration knowledge [12]. MBSE follows a

complementary direction by integrating requirements, structures, behaviors, properties, and relationships within a system model [3, 13]. For the proposed approach, we therefore require an explicit allocation of information authority between the system model and external configuration mechanisms.

Several studies connect SysML modeling and configuration knowledge. Albers et al. apply MBSE to support modular design and the consistent representation of modular product architectures [6]. Hanna et al. link methodical modularization, product configuration, and an effect model [7]. Rigger et al. formalize configuration structures and dependencies and transform them into an executable configuration model [14]. These SysML v1-based approaches serve as a foundation for the current organization of configuration-relevant information using SysML v2 concepts.

SysML v2 distinguishes reusable definitions from contextual usages and provides specialization, redefinition, multiplicities, constraints, and variation elements [8]. Epp et al. propose a metamodel for conceptual and asset variability [15]. Boelsen et al. provide modeling guidelines for reusable mechanical elements [16]. Building on these contributions, this paper examines their integration with explicit configuration criteria and links to project-specific configurations and physical individuals.

Recent approaches use SysML v2 for automated architecture exploration. Allegaert et al. represent architectural variability, connectivity constraints, and design-space definitions [17], while Bussemaker et al. connect a SysML v2 extension to an external optimization environment [18]. These contributions motivate assessable configuration information; the present study focuses on its organization within modular system models.

Model-based modular systems and requirement-oriented configuration have been demonstrated for sensor-integrating machine elements [19, 20]. These works provide the application basis for the present SysML v2 modeling approach.

The reviewed literature provides concepts for modular development, variability, configuration knowledge, and reusable system modeling. This paper synthesizes these concepts into operational assessment criteria and a four-level modeling structure. It investigates how reusable definitions, architecture-specific roles, technical variants, constraints, project configurations, and physical individuals can be related to support configuration.

This leads to the following research question:

Which modeling requirements should a modular SysML v2 system model satisfy to provide the information required for the computational derivation, validity assessment, and traceability of system configurations?

3. Methodology

To answer the research question, the study follows a three-stage procedure comprising requirement derivation, operationalization, and preliminary assessment.

First, modeling requirements are synthesized from modular product architecture, knowledge-based configuration, Product Line Engineering, and MBSE with SysML v2. Based on the focused review in Section 2, the synthesis identifies the information required to describe modular solution spaces, restricting combinations, derive project-specific configurations, and assess their validity. The review does not claim exhaustive coverage. Related concepts are consolidated into six assessable modeling requirements and linked to their literature basis.

Second, the requirements are operationalized through a four-level modeling approach. Each level is assigned a distinct semantic responsibility, and the resulting concepts are mapped to corresponding SysML v2 language elements.

Third, the approach is applied to a focused example involving a sensor-integrating bolt. Partial modeling in Cameo Systems Modeler informed the example; assessment is limited to

manual inspection of selection and extension scenarios. Complete tool validation, API-based access, industrial scalability, and optimization are outside the demonstrated scope.

4. Requirements for Configuration-Ready System Models

The following requirements result from the synthesis described in Section 3 and operationalize configuration readiness for modular SysML v2 system models.

4.1. Operational Definition of Configuration Readiness

A modular system model is considered configuration-ready if it provides an explicit, internally consistent, and machine-accessible representation of the modular solution space and the information required to derive, assess, and trace project-specific system configurations.

Configuration readiness is a property of the represented information, not of the configuration algorithm. The model does not have to select or optimize solutions, but must make system elements, variation points, admissible alternatives, requirements, and constraints accessible to subsequent configuration processes.

4.2. Derived Modeling Requirements

Table 1 summarizes six requirements synthesized for this study. The references support individual aspects; their grouping and operational formulation define the proposed assessment criteria.

Table 1: List of modeling requirements for configuration-ready system models

ID	Modeling requirement	Operational meaning	Literature basis
R1	Explicit modeling semantics and identity	The model shall distinguish reusable definitions, contextual usages, selectable variants, project-specific configurations, and physical individuals. All configuration-relevant elements shall be uniquely identifiable.	SysML v2 definitions and usages; variability and reuse modeling, and identity [8, 15, 16, 21]
R2	Hierarchical modular architecture and explicit interfaces	The model shall represent systems, modules, components, recursive compositions, multiplicities, typed properties, and configuration-relevant interfaces.	Modular architecture and SysML v2 structure [1, 2, 6, 7, 8, 9]
R3	Explicit variability and constrained configuration space	The model shall identify variation points and their admissible alternatives. Cardinalities, compatibility conditions, requires/excludes relationships, and constraints shall restrict the valid solution space.	PLE, knowledge-based configuration, and SysML v2 variability [4, 5, 8, 10, 11, 14, 15]
R4	Requirements-driven configuration and traceability	Project-specific requirements shall be parameterizable and traceable to affected architecture elements, properties, configuration decisions, analyses, and verification evidence.	MBPLE, SysML v2 requirements, and application studies [5, 7, 8, 19, 20]
R5	Assessable configuration completeness and validity	The represented information shall enable a configuration process to distinguish incomplete, complete-invalid, and complete-valid configurations. Selected elements, parameter values, aggregated properties, and constraints shall be assessable.	Configuration knowledge, formalization, and architecture generation [10, 11, 14, 17, 18]

R6	**Machine accessibility, authority, and explainability**	**Configuration-relevant information shall be accessible through stable identifiers and documented machine interfaces. For redundant or derived representations, the authoritative source and provenance shall be defined. Configuration decisions and violations shall be traceable to the originating model elements and constraints.**	**API access, configuration documentation, and formalization [12, 14, 21]**

R1–R6 cover the semantic and structural model foundation, the bounded solution space, requirements traceability, configuration assessment, and machine access. They are proposed criteria within the scope of this study for a configuration-ready modular system model without prescribing a specific tool, repository structure, or configuration algorithm.

5. Configuration-Ready Modeling Approach

To operationalize the modeling requirements introduced in Section 4, the proposed approach separates configuration-relevant information across four related modeling levels. The levels represent distinct semantic responsibilities rather than a prescribed file or repository structure. The relationships between the levels connect reusable engineering knowledge with architecture-specific variability, project-specific configurations, and physical individuals.

5.1. Four-Level Modeling Architecture

Configuration-relevant information fulfills different purposes within a modular system model. Domain concepts provide the common terminology, the modular architecture defines the required structural roles, technical variants provide possible solutions for these roles, and project configurations document concrete selections. Separating these purposes prevents architecture roles, available alternatives, selected elements, and physical objects from being treated as equivalent.

The proposed architecture therefore distinguishes four levels, progressing from general domain definition to project-specific system solutions. Figure 1 illustrates the levels and their relationships using the example of a sensor-integrating bolt.

Level 1 – Domain and Language Foundation provides the common vocabulary and modeling conventions used at the subsequent levels. It defines general concepts such as components, modules, interfaces, and units, as well as domain-specific terminology. For the example, this level provides the abstract module types required to describe sensor-integrating machine elements.

Level 2 – Modular System Architecture describes the reusable structure of the system family. It defines structural elements, architecture roles, compositions, multiplicities, and interfaces independently of concrete solutions. Roles at which different solutions may be used represent configuration decisions (variation points) within the architecture. In the bolt example, roles include sensing, energy supply, and data processing. These roles can be satisfied by specialized kind of modules (variation points) of defined multiplicity.

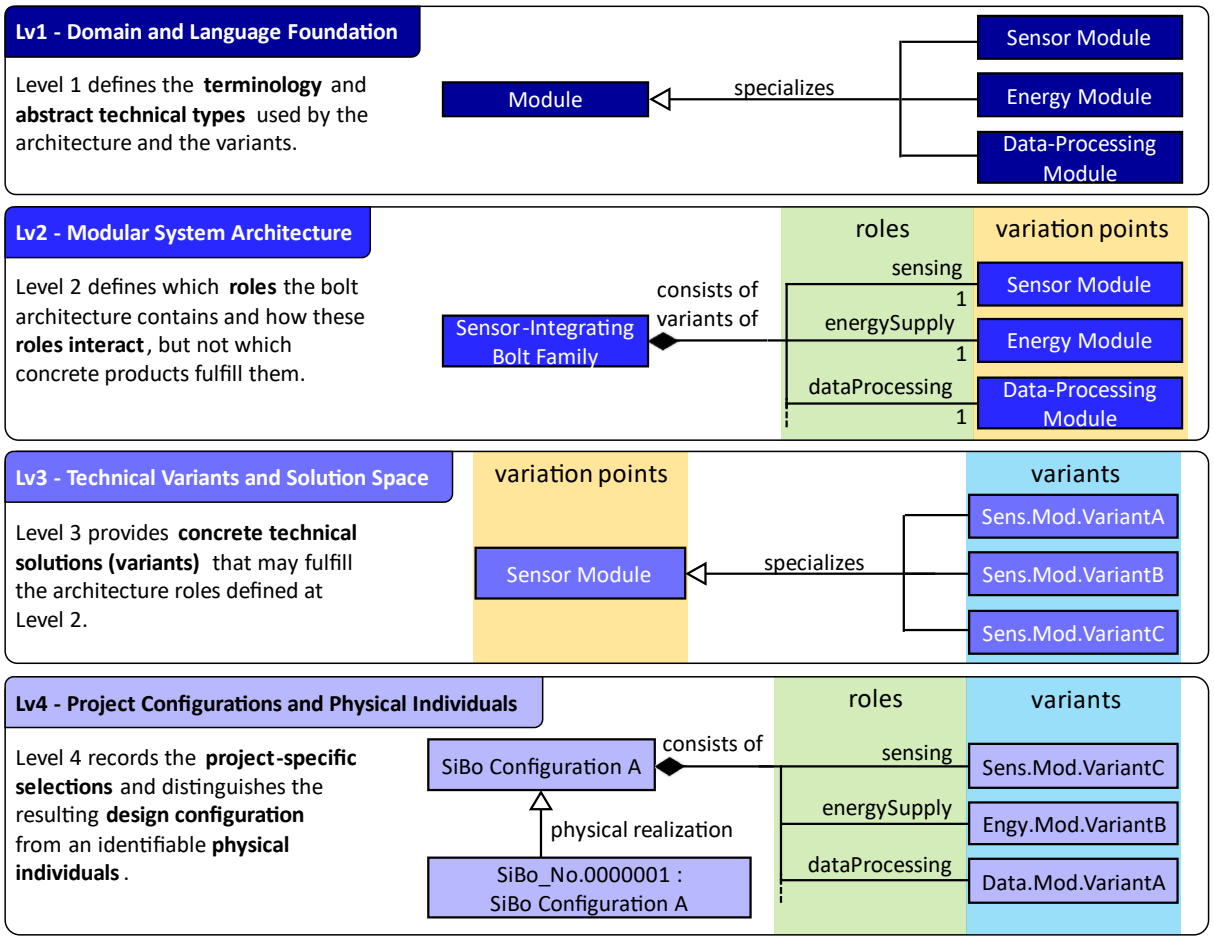

Figure 1: Four-level modeling architecture for configuration-ready system models, illustrated by a sensor-integrating bolt (SiBo).

Level 3 – Technical Variants and Solution Space contains reusable technical solutions that can be placed in the variation points to fulfill the architecture roles defined at Level 2. These variants describe concrete technical properties, interfaces, internal compositions, and local constraints. In the example, different sensor variants may be provided.

Level 4 – Project Configurations and Physical Individuals represents the results derived from the modular architecture and the available technical variants. A design configuration contains selected variants with parameter values and relates them to project-specific requirements. A physical individual additionally represents a uniquely identifiable realized object and may include serial numbers and test results. The distinction prevents a reusable design configuration from being confused with a particular manufactured exemplar.

Level 1 provides the domain semantics used to define the architecture at Level 2. Level 3 provides technical variants for its variation points, while Level 4 records their project-specific selection and realization. Constraints may act within individual levels or across them, for example by restricting variant assignments and combinations.

5.2. SysML v2 Representation and Cross-Level Consistency

Table 2 maps the concepts of the four-level approach to native SysML v2 concepts. It distinguishes technical types, architecture roles, reusable variants, and project-specific selections.

Table 2: Mapping of approach concepts with SysML v2 representations

Concept of the approach	SysML v2 representation	Semantic purpose
Abstract component or module type	*abstract part def*	Reusable technical category; used for domain foundation
Modular system architecture	*part def* containing interconnected *part usages*	Reusable structure of the system family
Architecture role	*part usage*	Context-specific position within the architecture
Variation point	*variation part usage*	Architecture role requiring a configuration decision
Technical variant	specialized *part def*	Reusable technical solution
Admissible alternative	*variant part usage* typed by a catalog definition	Candidate permitted at a variation point
Property	typed *attribute usage*	Machine-interpretable technical value
Interface	*port def* and *port usage*	Technical compatibility information
Configuration rule	*constraint def* and *constraint usage*	Restriction of admissible combinations
Reusable requirement	*requirement def*	Parameterizable requirement template
Project requirement	*requirement usage*	Project-specific parameter assignment and subject
Design configuration	*part def*	Fully selected technical solution
Physical exemplar	*individual part def*	Uniquely identifiable real object

A technical variant is represented as a specialized part def at Level 3. Its admissibility for a Level 2 architecture role is expressed by a variant part usage within the corresponding variation point, typed by or referring to the reusable variant definition. This allows technical variants to be reused across architecture roles and system families.

At Level 4, a configuration specializes the modular architecture and redefines its variation points by assigning selected variants. It is complete when all mandatory variation points and required parameter values are resolved. Project-specific requirement usages bind requirements to the assessed configuration, while a physical individual adds identifying and as-built information.

Cross-level consistency is established through typing, specialization, redefinition, multiplicities, interfaces, and constraints. Selected variants must conform to their architecture roles and satisfy applicable compatibility and configuration rules. Constraints may act within or across levels, including the assignment of variants to roles and the assessment of configurations against project requirements.

When external databases or configurators are used, the SysML v2 model is intended to remain authoritative for architecture, variants, requirements, and constraints. Operational representations and proposed changes must retain references to their originating model elements and be validated before incorporation into the authoritative model.

6. Demonstration and Preliminary Assessment

The sensor-integrating bolt example is inspected against R1–R6 to assess the information represented for configuration decisions. The assessment is limited to representative model elements and selected configuration scenarios; it does not demonstrate an executable workflow.

6.1. Demonstrator and Model Scope

The example illustrates the four modeling levels introduced in Figure 1 with the SysML v2 representations summarized in Table 2. Figure 2 presents schematic excerpts. Omitted definitions and constraint bodies are not executable evidence.

Level 1 contains abstract domain definitions for sensor-integrating machine elements and the required module types. Level 2 defines the modular bolt architecture through roles for sensing, energy supply, data processing, and communication. To limit the scope of the example, only the sensing role is represented as a variation point. Level 3 provides a strain-gauge sensor and a temperature sensor as reusable technical variants and defines them as admissible alternatives for this role. Level 4 represents a project-specific bolt configuration selecting the strain-gauge sensor and a corresponding physical individual.

The demonstrator is intentionally restricted to the elements required to illustrate the separation and relationships between domain definitions, architecture roles, technical variants, project configurations, and physical individuals. It does not represent a complete technical design or the full configuration space of a sensor-integrating bolt.

Level 1 – Domain Foundation

```
abstract part def
SensorIntegratingMachineElement;
port def DataPort;
abstract part def SensorModule {
  attribute energyDemand : Real;
  port dataOut : DataPort;
}
```

Level 2 – Modular Architecture

```
part def SensorIntegratingBolt
    :>
SensorIntegratingMachineElement {
  variation part sensing :
SensorChoices [1];
    part energySupply :
EnergyModule[1];
    assert constraint
energyBalance { ... }
}
```

Level 3 – Technical Variants

```
part def StrainGaugeSensor :>
SensorModule;
part def TemperatureSensor :>
SensorModule;
variation part def SensorChoices
:> SensorModule {
  variant part strainGaugeSensor
    : StrainGaugeSensor;

  variant part temperatureSensor
    : TemperatureSensor;
}
```

Level 4 – Configuration and Individual

```
part def BoltConfigurationA :>
SensorIntegratingBolt {
  attribute configurationId =
  "CFG-BOLT-001";
  attribute sourceModelVersion =
  "v1.0";
  part :>> sensing =
  sensing::strainGaugeSensor;
}
requirement energyAutonomy { ... }
individual part def bolt_SN_001
  :> BoltConfigurationA;
```

Figure 2: Exemplary SysML v2 excerpts for the sensor-integrating bolt.

6.2. Scenario-Based Preliminary Assessment

Three scenarios guide the inspection. In the valid scenario (see Figure 2 *Level 4*), the *strainGaugeSensor* is selected as an admissible alternative for the *sensing* role. The selection of a sensor outside *SensorChoices* (Level3) leads to an unvalid scenario. The third scenario is an extention. Through an additional *variant part,* the number of possible configurations can be increased. These scenarios illustrate intended behavior.

The model partly shown in figure 2 is assessed against R1–R6. A requirement is classified as demonstrated if the required information is explicitly represented and illustrated, and as partially demonstrated if only selected aspects are covered or manual interpretation remains necessary. The results in Table 3 are abbreviated as D (demonstrated), PD (partially demonstrated), and N (not demonstrated).

Table 3: Preliminarily assessment of the demonstrator against the modeling requirements

Requirement	Evidence in the demonstrator	Result
R1	Definitions, usages, roles, variants, configurations, and an individual are represented separately.	D
R2	Composition, multiplicity, a typed property, and a port are shown; recursive decomposition and interface connections are not detailed.	PD
R3	A variation point, alternatives, and selection cardinality are explicit; compatibility and configuration constraints restrict the solution space.	D
R4	A requirement placeholder is shown; parameterization and traceability are not demonstrated.	N
R5	Incomplete, complete-invalid, and complete-valid configurations can be distinguished; constraint evaluation is not validated	PD
R6	Configuration and version identifiers are illustrated; API access and rule-level provenance are untested.	PD

R1 and R3 are demonstrated within the illustrated example through semantic separation and identifiable model elements. R2, R5, and R6 are partially covered; R4 lacks demonstrated parameterization and traceability. This supports the proposed information organization, but does not establish executable configuration readiness or general applicability.

7. Conclusion and Outlook

This paper investigated which information a modular SysML v2 system model must provide to support the computational derivation, validity assessment, and traceability of system configurations. Based on a concept-oriented synthesis of modular product architecture, knowledge-based configuration, Product Line Engineering, and MBSE with SysML v2, six modeling requirements for configuration-ready system models were derived. Configuration readiness was defined as a property of the represented information.

The requirements were mapped to four levels: domain foundations, modular system architecture, solution space with technical variants, and project-specific configurations including physical individuals. Their mapping to SysML v2 concepts distinguishes reusable definitions, architecture roles, variation points, admissible technical variants, concrete selections, and individuals. The focused application to a sensor-integrating bolt example demonstrates semantic separation and its effects. Requirement traceability, complete constraints, and API access remain unverified. The contribution is a literature-informed organization of SysML v2 concepts and assessment criteria, rather than a validated configuration-ready modular system.

The study is limited to a focused literature synthesis and illustrative scenarios. Future work should validate a complete SysML v2 implementation, demonstrate requirement traceability and configuration assessment, and implement API access, provenance, and controlled synchronization. Applicability to other modular systems and industrial scalability remain open.

Acknowledgments

This research was funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Project-ID 551485258 as part of the SPP2305 (Project-ID 441853410). The responsibility for the content of the report lies with the authors.

References

[1] Simpson, Timothy W. et al. (Hrsg.): Product Platform and Product Family Design: Methods and Applications. New York: Springer, 2006. DOI: 10.1007/0-387-29197-0.

[2] Krause, Dieter; Gebhardt, Nicolas: Methodische Entwicklung modularer Produktfamilien: Hohe Produktvielfalt beherrschbar entwickeln. Berlin, Heidelberg: Springer, 2018. DOI:10.1007/978-3-662-53040-5.

[3] Friedenthal, Sanford; Moore, Alan; Steiner, Rick: A Practical Guide to SysML: The Systems Modeling Language. 3rd Edition. Waltham, MA, USA: Elsevier, 2015. DOI: 10.1016/C2013-0-14457-1.

[4] Pohl, Klaus; Böckle, Günter; van der Linden, Frank: Software Product Line Engineering: Foundations, Principles, and Techniques. Berlin, Heidelberg: Springer, 2005. DOI: 10.1007/3-540-28901-1.

[5] Forlingieri Marco, Weilkiens Tim, Chalé-Gongora Hugo G.: Model-Based Product Line Engineering (MBPLE): The Feature-Based Path to Product Lines Success. Wiley, 2025. DOI: 10.1002/9781394204694.

[6] Albers, Albert et al.: “Model-based systems engineering in modular design.” Design Science, Vol. 5, e17, 2019. DOI: 10.1017/dsj.2019.15.

[7] Hanna, Michael et al.: “A Model-Based Approach for the Methodical Development and Configuration of Modular Product Families.” Systems, Vol. 11, No. 9, 449, 2023. DOI: 10.3390/systems11090449.

[8] Object Management Group: Systems Modeling Language, Version 2.0, Part 1: Language Specification. March 2026. OMG formal/2026-03-02. Available at: https://www.omg.org/spec/SysML/2.0/Language/PDF.

[9] Ulrich, Karl T.: The Role of Product Architecture in the Manufacturing Firm. In: Research Policy 24 (1995), Nr. 3, S. 419–440. DOI: 10.1016/0048-7333(94)00775-3.

[10] Hvam, Lars; Mortensen, Niels Henrik; Riis, Jesper: Product Customization. Berlin, Heidelberg: Springer, 2008. DOI: 10.1007/978-3-540-71449-1.

[11] Felfernig, Alexander et al. (Hrsg.): Knowledge-Based Configuration: From Research to Business Cases. Waltham: Morgan Kaufmann, 2014. ISBN-13: 978-0-12-415817-7.

[12] Shafiee, Sara et al.: The documentation of product configuration systems: A framework and an IT solution. In: Advanced Engineering Informatics 32 (2017), S. 163–175. DOI: 10.1016/j.aei.2017.02.004.

[13] Delligatti, Lenny: SysML distilled: a brief guide to the systems modeling language. Upper Saddle River, NJ: Addison-Wesley; 2014. ISBN-13: 978-0321927866.

[14] Rigger, Eugen; Fleisch, Ruth; Stankovic, Tino: Facilitating Configuration Model Formalization based on Systems Engineering. In: Proceedings of the 23rd International Configuration Workshop. CEUR Workshop Proceedings 2945 (2021), S. 1–8. Available at: https://ceur-ws.org/Vol-2945/11-ER-ConfWS21_paper_14.pdf.

[15] Epp, Jordan et al.: Transitioning towards SysML v2 as a variability modeling language. In: Innovations in Systems and Software Engineering 20 (2024), S. 585–596. DOI: 10.1007/s11334-024-00569-y.

[16] Boelsen, Kathrin et al.: SysML v2 based modelling guidelines for mechanical system elements. In: Forschung im Ingenieurwesen 89 (2025), Nr. 1, Art. 60. DOI: 10.1007/s10010-025-00827-w.

[17] Allegaert, Elias et al.: Formulating System Architecture Generation Problems Using SysML v2. In: 2026 IEEE International Systems Conference. Piscataway: IEEE, 2026. DOI: 10.1109/SysCon66367.2026.11503624.

[18] Bussemaker, Jasper et al.: System Architecture Optimization Using SysML v2: Language Extension and Implementation. In: 2026 IEEE International Systems Conference. Piscataway: IEEE, 2026. DOI: 10.1109/SysCon66367.2026.11503593.

[19] Küchenhof, Jan et al.: Microelectronic modular system for sensor-integrating machine elements—model-based configuration and prototypical implementation of sensor-integrating bolts. In: Forschung im Ingenieurwesen 89 (2025), Nr. 1, Art. 164. DOI: 10.1007/s10010-025-00931-x.

[20] Küchenhof, Jan et al.: Development of a model-based modular building kit for sensor-integrating machine elements—Theory and application. In: Forschung im Ingenieurwesen 88 (2024), Art. 40. DOI: 10.1007/s10010-024-00761-3.

[21] Object Management Group: Systems Modeling Application Programming Interface (API) and Services, Version 1.0. March 2026. OMG formal/2026-03-04. Available at: https://www.omg.org/spec/SystemsModelingAPI/1.0/PDF.