Article

Not peer-reviewed version

A Unified Specification Process for Graphical Domain-Specific Languages in Model-Based Systems Engineering

Katharina Polanec *, Simon Eschlberger, Markus Michael Peter, David Hoffmann, Arndt Lüder, Christian Neureiter

Posted Date: 12 May 2026

doi: 10.20944/preprints202605.0788.v1

Keywords: Domain-Specific Language (DSL); Model-Based Systems Engineering (MBSE); metamodeling; ontology-based language specification

 doi:10.20944/preprints202605.0788.v1

Article

A Unified Specification Process for Graphical Domain-Specific Languages in Model-Based Systems Engineering

Katharina Polanec [1,2,*], Simon Eschlberger [1], Markus Michael Peter [1], David Hoffmann [2], Arndt Lüder [2] and Christian Neureiter [1]

[1] Center for Dependable Systems Engineering, Salzburg University of Applied Sciences, Urstein Sued 1, 5412 Puch/Salzburg, Austria
[2] Institute for Engineering of Products and Systems (IEPS), Otto-v.-Guericke University, Universitätsplatz 2, 39106 Magdeburg, Germany
* Correspondence: katharina.polanec@fh-salzburg.ac.at

Abstract

Rising complexity in cyber-physical systems development exposes challenges in the consistent and reusable specification of graphical domain-specific languages (DSLs). Despite the benefits of model-based systems engineering (MBSE), the absence of a standardized, lifecycle-wide specification process results in semantic inconsistencies, tool dependence, and limited interoperability. While our previous work has addressed individual stages of DSL definition, a comprehensive, standards-based process integrating these stages remains missing. Building on these foundations, this paper introduces a unified language specification process for graphical DSLs grounded in established standards—the Meta-Object Facility (MOF), Unified Modeling Language (UML), Web Ontology Language (OWL), and Resource Description Framework (RDF). The process integrates three core artifacts: a tool-independent ontology capturing domain semantics, a MOF-conformant metamodel unifying abstract syntax, semantics, and concrete syntax, and a UML-profile-based implementation. To support and exemplify this process, a prototypical toolchain is introduced that enables automated transformations between these artifacts, thereby facilitating the consistent propagation of semantics from ontology to implementation. The applicability of the proposed process is demonstrated through both a top-down automotive case and a bottom-up cybersecurity DSL, illustrating its cross-domain generalizability. By explicitly structuring and connecting ontology, metamodel, and implementation, this work contributes a semantically consistent, machine-interpretable, and tool-independent specification process for graphical DSLs in MBSE.

Keywords: Domain-Specific Language (DSL); Model-Based Systems Engineering (MBSE); metamodeling; ontology-based language specification

1. Introduction

Developing modern cyber-physical systems (CPSs) poses significant challenges. Due to their raising complexity, it becomes increasingly important to integrate heterogeneous components and explicitly consider interfaces and interrelations in all directions. Therefore, the utilization of models, computer-aided engineering, and design methodologies is becoming essential [1].

Moreover, the development of CPSs requires interdisciplinary knowledge integration and continuous exchange of digital artifacts [2] throughout all lifecycle phases [3]. However, collaboration is often hindered by inefficient communication and inconsistent information handling, leading to costly delays, quality issues, and limited stakeholder acceptance [4].

Systems engineering—a well-established and integrative lifecycle approach for engineering complex systems—is frequently applied to manage complexity [5]. Combined with model-based

 doi:10.20944/preprints202605.0788.v1

methods, it has evolved into model-based systems engineering (MBSE), a validated strategy in sensitive domains like aerospace and defense [5]. However, to ensure effective communication and collaboration between stakeholders from diverse professional backgrounds, models must be tailored to their concerns and application domain [4].

Therefore, domain-specific languages (DSLs) have emerged as promising tools for model-based approaches, precisely tailored to specific application domains without sacrificing the precision and unambiguity of modeling [6]. They bridge the gap between technical precision and stakeholder comprehension. Nevertheless, practical experience gained through research collaborations with industry partners reveals that even within individual organizations, inconsistencies in fundamental terminology remain, resulting in communication challenges across the lifecycle and beyond the scope of MBSE.

While MBSE and DSLs provide effective means to structure system models, DSL-related concepts and terminology are not limited to modeling environments alone. They must often be consistently applied across heterogeneous lifecycle artifacts and tools. This highlights the need for more formalized and standardized specification approaches that ensure semantic consistency beyond individual modeling environments.

Despite growing DSL adoption in model-based approaches, their engineering still lacks a flexible, widely-used, and standardized specification process [7]—especially regarding their syntax and semantics. This leads to

- proprietary DSL solutions,
- high maintenance efforts,
- inconsistent documentation,
- tool-dependent implementations,
- limited reusability and interoperability,

ultimately resulting in reduced stakeholder acceptance.

Several domains have addressed these challenges by introducing domain-specific architecture frameworks in combination with graphical DSLs. For example, the Smart Grid Architecture Model (SGAM) in energy [8] and Reference Architecture Model Industrie 4.0 (RAMI 4.0) in industrial automation [9]. Both frameworks were made applicable in previous work from our research group with dedicated DSLs using Unified Modeling Language (UML) profiles [10,11].

Building on these foundations, Polanec et al. [12] introduced the Automotive Reference Architecture Model (ARAM) for the automotive domain, including a graphical DSL and modeling tool support, the ARAM Toolbox [12]. However, the implementation and engineering of the ARAM DSL once more revealed the lack of a standardized DSL specification process—no formal blueprint existed despite lessons learned from SGAM and RAMI 4.0.

Existing standards partially address this gap. Languages like UML use metamodels for abstract syntax and semantics [13–15], with Meta-Object Facility (MOF) as meta-metamodel [16]. However, MOF lacks a standardized definition of the language's visual representation, the concrete syntax. Therefore, graphical definitions are done in separate artifacts, resulting in inconsistencies, or not done at all, which poses a challenge for maintaining and adapting a customized modeling language.

Ontology-based approaches, in turn, offer tool-independent vocabulary and semantic grounding for domain concepts [17], but debates persist on whether to merge ontology and metamodel into one artifact [18] or keep them separate [19,20]. This ongoing divergence highlights the need for a unified specification process.

Against this background, our research began with the goal of establishing a formalized way of specifying graphical DSLs implemented as UML profiles. The initial focus was on the ARAM DSL, where the lack of standardized guidance became evident.

As a first step, we proposed a MOF-based specification approach which integrates abstract syntax, semantics, and concrete syntax into a unified metamodel artifact [21]. This approach reduces

 doi:10.20944/preprints202605.0788.v1

inconsistencies and simplifies DSL engineering by providing a single source of truth for the modeling-language definition.

Thereafter, we recognized the need for a more generic approach towards formalizing languages, since the UML profile mechanism is limited to MBSE environments and does not cover the entire lifecycle of a CPS. To address this, we proposed an ontology-driven specification process for graphical DSLs, using ontologies as a formal, tool-independent vocabulary of domain concepts and their relations [22]. Ontologies are widely used in various domains to capture knowledge and enable interoperability between different systems. By using ontologies as the semantic foundation for DSL engineering, we can ensure consistency and interoperability across different lifecycle stages and tools. In this process, we also compared the metamodel and ontology artifacts, clarifying their differences and complementary strengths [22].

To ensure consistency between ontology, metamodel, and language implementation, we also developed prototypes that automatically transform MOF-conformant metamodels into UML profiles [23] as well as ontologies defined in Web Ontology Language (OWL) into MOF-conformant metamodels [24].

Altogether, these research steps gradually evolved from a domain-specific architecture framework toward a holistic, formalized, and automated language specification process. The result is a prototypical toolchain that spans from ontology to metamodel to concrete tool implementation, supporting reusable, semantically consistent, and machine-interpretable DSLs across the entire system lifecycle.

In this work, we build on our previously published research steps and consolidate them into a coherent language specification process for graphical DSLs. While earlier work addressed individual stages of DSL definition, this paper explicitly integrates these stages into a unified, standards-based process that aligns ontology-based semantics, MOF-conformant metamodeling, and UML-profile-based implementations. We thereby capture the overall research journey, which originated from the ARAM DSL as a concrete application in the automotive domain and was subsequently generalized toward cross-domain interoperability, semantic consistency, and tool-independent DSL engineering. Based on this foundation, we address the following leading research question:

> *How can the specification of model-based languages be formalized in a way that ensures semantic consistency, reuse, and machine-interpretability while remaining extensible toward non-modeling contexts?*

To answer this research question, this paper makes the following contributions:

1. We introduce a unified, standards-conformant specification process that explicitly connects ontology, metamodel, and implementation artifacts, thereby enabling a consistent and reusable definition of graphical DSLs.
2. We consolidate previously introduced transformation approaches into an integrated toolchain that extends the proposed process toward automation. The toolchain serves to demonstrate the feasibility of automated artifact transformations and supports the consistent propagation of semantics across ontology, metamodel, and implementation.
3. Finally, we demonstrate the applicability and cross-domain generalizability of the proposed process through its application to a cybersecurity DSL case study [25].

Altogether, this paper consolidates our previous stepwise contributions into a unified language specification process and corresponding tool-supported transformation chain. The approach is exemplified in the cybersecurity domain and complemented by a third prototype, resulting in an integrated toolchain for ontology-to-implementation graphical DSL engineering.

The remainder of this paper is structured as follows: Section 2 outlines the theoretical background and related work. Section 3 presents the research approach, scope, applied standards, and tools, as well as the derived requirements. Section 4 details the development of the language implementation, metamodel, and ontology, while Section 5 demonstrates their application. Section 6 evaluates the approach through a prototypical implementation, and Section 7 concludes the paper.

 doi:10.20944/preprints202605.0788.v1

2. Background

Before detailing our approach, we first provide background on MBSE, domain-specific systems engineering (DSSE), metamodeling with MOF, and ontologies as a semantic foundation. We then review related work to position our contribution in the broader research landscape.

2.1. From MBSE towards Domain-Specific Systems Engineering

The increasing complexity of CPSs and modern systems in general increases the need for interdisciplinary collaboration across the life cycle. While systems engineering provides a structured, interdisciplinary approach across the whole life cycle [5,26], MBSE further emphasizes formalized modeling practices to manage complexity and assure consistency and traceability throughout [27]. Although empirical evidence of the benefits of MBSE is limited, it is recognized that many benefits stated by institutions like International Council on Systems Engineering (INCOSE) are widely observed by practitioners [28]. The typically used modeling language Systems Modeling Language (SysML) [29] is valuable in this context, yet its technically oriented nature makes it difficult to apply for stakeholders from other technical but non-specialist, or even non-technical backgrounds. Current developments toward SysML 2.0 [30] might improve the usability and expressiveness of models, however, due to the novelty of the language, tool support is still limited and practical experience is scarce.

To overcome these limitations and place stronger focus on stakeholder involvement, DSSE has been proposed [31]. DSSE combines the strengths of MBSE with modeling languages tailored to the needs of specific domains, enabling both precision and accessibility. This extension of MBSE also introduces a modeling stack, where an analysis model—expressed in a DSL—serves as the entry point, and an architectural model provides the technical detail in SysML. This modeling stack adds a stakeholder-oriented layer on top of the technically oriented SysML diagrams while still maintaining traceability and consistency throughout the system architecture [31].

DSSE has been successfully demonstrated in several domains. The SGAM [8] and RAMI 4.0 [9] each provide standardized frameworks, accompanied by graphical DSLs and supporting toolboxes [10,11]. Inspired by these efforts, the ARAM framework was introduced for the automotive domain [12]. Unlike SGAM and RAMI 4.0, the ARAM-based DSL was developed with a formally defined MOF-based metamodel, comprising both abstract syntax and semantics. However, the absence of a standardized definition for the concrete syntax exposes challenges in maintaining usability, interoperability, and formal rigor—an issue that motivates the research addressed in this paper.

2.2. Metamodels and the Role of MOF

In DSL development, we distinguish between *language specification* and *language implementation* [21]. The specification defines a modeling language's underlying structure and meaning independently of its realization. The implementation on the other hand can vary—for instance as a UML profile or as a simple graph structure—without altering the fundamental concepts. Notably, the implementation is typically tool-dependent whereas the specification should be as tool-independent as possible to ensure effective reusability. Following Kleppe [13], metamodels—models that specify other models—provide the formal basis for language specification and typically comprise three parts: the abstract syntax model (ASM), the concrete syntax model (CSM), and transformations between them. The ASM defines the constructs and their valid relations, in other words allowed nodes and valid edges, while the CSM defines their representation, or how elements are made accessible to human senses. One ASM may admit multiple CSMs, as in UML which can be displayed as a graphical diagram or serialized as XML Metadata Interchange (XMI). Although formal languages do not strictly require explicit semantics, a semantic model (SM) is essential for DSLs with practical relevance [21]. In short: ASM defines structure, CSM defines representation, and SM defines meaning of a language [13].

A fundamental standard for metamodel definition, especially in the context of MBSE, UML, and SysML, is the MOF, which serves as a meta-metamodel [16]. MOF defines a restricted subset of UML class-diagram elements and primarily standardizes how a language's abstract syntax can

 doi:10.20944/preprints202605.0788.v1

be defined. Semantics are typically attached in the class element's notes compartment as a text in natural language. Therefore, the SM is implicitly defined by MOF, even though it is not explicitly standardized. However, the third integral part—the CSM—is not part of this standard. The Object Management Group (OMG) Diagram Definition (DD) [32] was introduced to cover concrete syntax; however, it has not been updated since its publication ten years ago, leaving its current relevance and acceptance unclear. Moreover, separating ASM and CSM results in two artifacts that must be maintained in parallel, which increases the risk of inconsistencies.

In practice, the absence of a holistic, standardized language-specification approach means many graphical DSLs lack a formal definition of their concrete syntax. This not only aggravates visual customization with for instance organization-specific icons for language elements but also fosters tool-specific, non-reusable implementations and hinders collaboration across organizations and domains. Gupta et al. [33] address this gap with a building-block approach for industrial DSLs. Complementary to that, our previous work proposed a MOF-based methodology that focuses on a standard-conformant definition of the graphical CSM and unifies ASM, SM, and CSM into a single, tool-independent metamodeling artifact [21]. The goal of this research was a machine-readable metamodel, that incorporates all necessary information about the modeling language, so that it remains accessible to domain experts, reduces maintenance effort, improves reuse, and enables automated generation of language implementations directly from the specification. The results of this research are summarized in Section 4.2.

2.3. Ontologies as a Semantic Foundation

When specifying a DSL for use across the entire life cycle—where not all phases are model-based—starting directly with metamodels can be premature. For instance, requirements engineering often relies on tools like Microsoft Excel rather than an MBSE tool, yet requirements and system architecture must still be semantically consistent. To ensure semantic consistency and reusability across all phases, the meaning of the language elements should first be stated explicitly, independent of the language's implementation [14]. An appropriate method to achieve that is the use of ontologies.

Ontologies are a central aspect to the Semantic Web where they are used to describe knowledge in a machine-interpretable form [34]. This knowledge is represented using the OWL as an ontology language [35] and the Resource Description Framework (RDF) [36].

The term ontology originates from philosophy and is particularly suitable to preserve the meaning of data when it is exchanged between systems or organizations. Ontologies describe the formal specification of a shared and interconnected vocabulary for a given domain, including classes, relationships, functions, and objects [17]. According to this definition, ontologies can serve as a single source of knowledge, providing a common understanding of domain concepts and their interrelations for any kinds of languages.

Since not every artifact along the system life cycle is a model, ontologies can help to ensure semantic consistency across all phases and artifacts. Accordingly, when a DSL must support model-based and non-model artifacts alike, the ontology should precede metamodeling: it provides the semantic basis for any subsequent metamodel or modeling-oriented DSL implementation, and it remains usable outside the MBSE context.

In literature, the relationship between ontology and metamodel is debated, as outlined in the next subsection. Our approach maintains a clear separation: the ontology defines semantics in a tool-independent way, and the metamodel specifies the modeling language structure. This allows the ontology to serve as a reusable, implementation-independent knowledge base that can also support lifecycle activities beyond modeling.

2.4. Related Work

The use of ontologies, metamodels, and DSLs in systems and software engineering as well as MBSE is well established. In the following, we review related work and highlight commonalities and differences with our approach. First we focus on the role of ontologies and metamodels as they are described in literature.

 doi:10.20944/preprints202605.0788.v1

Orellana and Mandrick [2] state that a reference ontology is essential for the systems engineering life cycle and propose a process-centric reference ontology based on ISO/IEC/IEEE 15288 [3]. They suggest that a layered ontological approach is most promising and assert that achieving an enterprise-level ontology is unlikely without incorporating a top-level systems engineering ontology. Similarly, Madni and Sievers [27] as well as Lu et al. [7] delineate the importance of ontologies in an MBSE to complement metamodels.

On the contrary, Hinkelmann et al. [18] argue that keeping them separate risks incompatible semantics. They therefore advocate for an ontology-based modeling approach in which a formal ontology both defines the semantics and acts as a metamodel, integrating ontology (semantics) with metamodel (abstract and concrete syntax). Ernadote [37] even uses the terms ontology and DSL synonymously.

Apart from these disjointed views on keeping ontology and metamodel separate or combined, other existing work proposes the use of ontologies as a foundation for specifying languages.

Walter et al. [15] introduced an ontology-based framework for DSLs in software modeling. They argue that DSLs consist of abstract syntax, semantics, and at least one concrete syntax, which aligns with our view and description of metamodel composition. Their contribution addresses the challenge that semantics and constraints are often not explicitly defined, by integrating OWL and Ecore into a new metamodeling language. While their work shares the motivation of formally grounding DSL semantics, their solution differs from ours: rather than combining ontology and metamodel into a new artifact, we deliberately separate them to ensure a tool- and implementation-independent knowledge base. Furthermore, while their context is software modeling, we focus on systems engineering and emphasize reuse, usability, and intuitiveness in addition to technical precision.

Earlier, Eriksson et al. [14] had already emphasized that metamodeling should be grounded in foundational ontologies and proposed to complement traditional modeling with a viewpoint on language use. Their argument supports our ontology-based language specification approach, particularly with respect to clarifying the relationship between semantics and metamodels.

In the context of systems engineering, van Ruijven [38] demonstrated the relevance of ontologies for effective information exchange across the lifecycle by introducing a systems engineering ontology. Although their specific ontology is not directly applicable to our work, their conclusion, that RDF-based graphs are a promising mechanism for data exchange in MBSE, reinforces our approach of utilizing ontologies as a basis for lifecycle integration.

Yang et al. [39] provide a comprehensive review of ontology-based systems engineering. They highlight ontologies as a means of improving knowledge management and traceability in systems engineering, and conclude that their role becomes increasingly important in model-based disciplines.

More recently, Wu et al. [40] proposed an ontology to support traceability in MBSE toolchains. Their focus is on stakeholder integration and lifecycle traceability, but unlike our approach, they do not address the specification and reuse of DSLs.

Taken together, existing work shows a broad recognition of ontologies as valuable for grounding modeling languages and supporting MBSE. Our contribution differs in its explicit separation of ontology and metamodel, strict adherence to standards, and emphasis on creating a reusable, implementation-independent foundation for graphical DSLs in systems engineering. Most importantly, rather than introducing a new ontology or metamodeling framework, we propose a language specification process that is grounded in established standards and practices. This makes our approach complementary to existing methods and fosters broader acceptance, as it requires no paradigm shift but demonstrates how ontologies, metamodels, and implementations can be combined in a structured and reusable way.

3. Research Design and Paradigm

Our research approach follows the design science research (DSR) paradigm [41]. This paradigm centers around the creation of artifacts to solve identified problems and is structured in three cycles: relevance, rigor, and design cycle. The relevance cycle connects DSR to the application domain

 doi:10.20944/preprints202605.0788.v1

by feeding in real-world requirements and returning evaluated artifacts to measure their utility in practice. The rigor cycle anchors the research in the scientific knowledge base, ensuring that the created artifacts are grounded in established theories, methods, and prior experiences. Finally, the design cycle iteratively constructs and evaluates the artifacts, refining them until they satisfy the requirements drawn from relevance and rigor. These cycles are displayed in Figure 1. Shapes displayed in dark green and text in white delineate the basis of DSR as presented in [41]. Rectangles highlighted in light green delineate the content of the present research mapped to the DSR paradigm. Each of the DSR cycles is represented by a section within this paper.

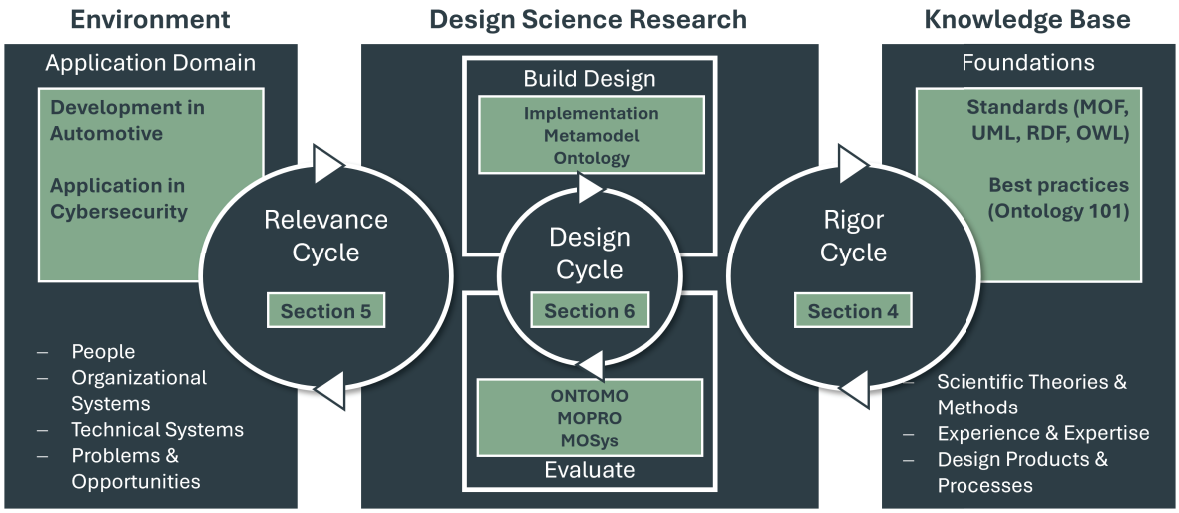

Figure 1. Structure according to Design Science Research (figure adapted and extended based on [41])

The present research endeavor addresses the problem of a missing standardized specification process for the development of graphical DSLs, which leads to inconsistencies, inefficiencies, and limited reusability. To tackle this challenge, we design a structured language specification process that comprises dedicated methods for the systematic development of ontology, metamodel, and implementation artifacts, as well as transformation mechanisms between them. Therefore, three specific methods are developed:

1. A method for developing a formal ontology, based on a well-established ontology engineering guide and realized using the RDF [36] and OWL [35] standards
2. A method for defining a metamodel, according to the MOF standard [16], capturing abstract syntax, semantics, and concrete syntax using UML class diagrams
3. A method for implementing the DSL as a UML profile [42], enabling its application within a modeling environment

These three methods, as listed in the build design within Figure 1, are developed based on established standards and best practices from literature. Each method produces a specific artifact, which is developed and evaluated using a representative example to assess the suitability of the chosen standards and methods. Following the evaluation results, the methods are adapted, and additional steps are defined within the range of the existing standards to fit the needs of the language specification process. This information flow from existing knowledge into the developed artifacts and the adapted standards and best practices from artifacts back to the knowledge base highlights the DSR rigor cycle and is described in detail in Section 4.

To answer our leading research question (introduced in Section 1), we split it into three research sub-questions (RQx), each addressing one of the three methods and their corresponding artifacts:

- **RQ1:** How can a formal ontology be developed to provide a tool-independent vocabulary and semantic grounding for domain concepts and their relations?

 doi:10.20944/preprints202605.0788.v1

- **RQ2:** How can a metamodel be defined to capture the abstract syntax, semantics, and especially concrete syntax of a graphical DSL in a standardized way?
- **RQ3:** How can a DSL be implemented as a UML profile to ensure stakeholder comprehensibility while maintaining technical precision?

To validate the cross-domain applicability of the introduced language specification process, we apply it to two different domains: the process is developed in a top-down manner demonstrated on the ARAM DSL [12] which is situated in the automotive domain (left side of the V in Figure 2) and then applied in a bottom-up manner to a cybersecurity DSL [25] (right side of the V in Figure 2). This transfer to another application domain represents the DSR relevance cycle and is described in Section 5.

Finally, to ensure and demonstrate semantic consistency between the artifacts created through the proposed methods, we develop two prototypes that are capable of transforming one artifact into another: Ontology-to-MOF Transformer (ONTOMO) [24] transforms an ontology into a metamodel, and MOF-based Profile Generator (MOPRO) [23] transforms a metamodel into a UML profile (DSR design cycle). Based on MOPRO we furthermore introduce a third prototype: MOF-based SysML-v2 Generator (MOSys) which is an adaptation of MOPRO to generate a second language implementation in the form of a SysML v2 library. These prototypes serve as evaluation to verify the semantic consistency and interoperability between the three main artifacts. Their development is described in Section 6 and represents the DSR design cycle.

In the following, we first outline the scope of this research as well as the standards and tools used. Thereafter, we describe the development of this language specification process and the interrelations between its development and application. We also address requirements for the artifacts and the developed prototypes.

3.1. Scope, Standards, and Tools

The scope of this research is the formalization of a specification process for high-level graphical DSLs, including their abstract syntax, semantics, and concrete syntax. This process is applied in the context of domain-specific modeling languages for the development of complex systems following a DSSE approach. To ensure broad applicability and interoperability, the research is based on established standards and best practices from literature, with focus on tool-independence to the greatest extent possible. Therefore, the ontology artifact is developed based on the RDF and OWL standards. For the development of the artifact itself we utilize the tool *WebProtégé* [43], which is the web application of the open source ontology editor *Protégé*. However, due to the standardized ontology formats, the ontology artifact is also compatible with other ontology editors capable of working with RDF and OWL.

The metamodel follows the MOF standard and utilizes UML. For the creation of this artifact we chose the modeling tool Enterprise Architect (EA) by Sparx Systems[1] due to our existing experience surrounding the developments of ARAM. EA is among the widely used modeling tools in the MBSE context [44] and supports XMI as underlying, standardized file format. While the presented metamodel in Section 4.2 is implemented within EA, its definition adheres strictly to MOF and UML standards. Therefore, the principles and structure of the metamodel can be reproduced with any UML-compliant modeling tool that supports MOF and XMI formats, ensuring tool independence of the underlying approach.

The DSL implementation is realized as a UML profile. Similarly to the metamodel, the profile mechanism itself is standardized by UML—and thus, in principle, tool-independent. However, the concrete, usable implementation inside a modeling environment is necessarily tool-dependent. In our case, we target EA and use its proprietary Model Driven Generation (MDG) Technology, which is an extension to the profile mechanism, to fully integrate the profile (modeling elements, diagrams, visualizations) into the tool. In other words: the specification basis (UML profile) is

[1] https://sparxsystems.com/products/ea/

 doi:10.20944/preprints202605.0788.v1

standard-conformant and portable, but the packaged implementation deliberately relies on EA's MDG format. We consider this tool-dependence appropriate, since an implementation is primarily realized for a specific tool.

As a result, the implementations of the first two prototypes—even though implemented as light-weight command-line applications programmed in C#—are in their current state tied to the modeling tool EA: ONTOMO uses the API of EA to generate the output directly within a modeling project. However, the content of it can be exported manually as standard-conformant XMI document in a following step. Following up, the MOPRO application generates the MDG Technology file based on this input. Both prototypes are implemented in a way that allows for easy adaptation to other tools and formats.

To verify the adaptability of the previously presented prototypes, in this work we introduce a third prototype—MOSys. It follows the same algorithm as MOPRO and utilizes the same metamodel artifact, but instead of a UML profile it generates a language implementation in the form of a SysML v2 library.

Alltogether, the choice of EA as the primary tool for prototype implementation is motivated by practical considerations, including existing expertise, tool maturity, and support for required standards (MOF, UML, XMI). While the implementations are currently tied to EA, the core principles, metamodel artifacts, and language definitions adhere strictly to open standards, ensuring that the conceptual approach remains tool-independent and portable. This approach balances the necessity of a concrete demonstration with the goal of general applicability; EA thus serves as a proof-of-concept environment, while the methodology itself remains fully based on open standards.

All artifacts and prototypes can be found in our public repository [2].

3.2. V-Model-Inspired Approach

The development of the language specification process follows a V-model-inspired approach, as visualized in Figure 2. The left side of the V represents the top-down development of the process using an example from the automotive domain. The development starts with the implementation of the ARAM-based DSL. This is followed by the definition of a formal metamodel, based on every aspect that is already known from the language implementation. Finally, the essence of the developed DSL is specified using an ontology which essentially encapsulates the domain understanding based on which the DSL was initially developed. At this stage, the fundamental core and meaning of the specified language is reached, therefore, the domain understanding is represented by the bottom tip of the V.

[2] https://github.com/K-Polanec/DSL-Specification-Process

 doi:10.20944/preprints202605.0788.v1

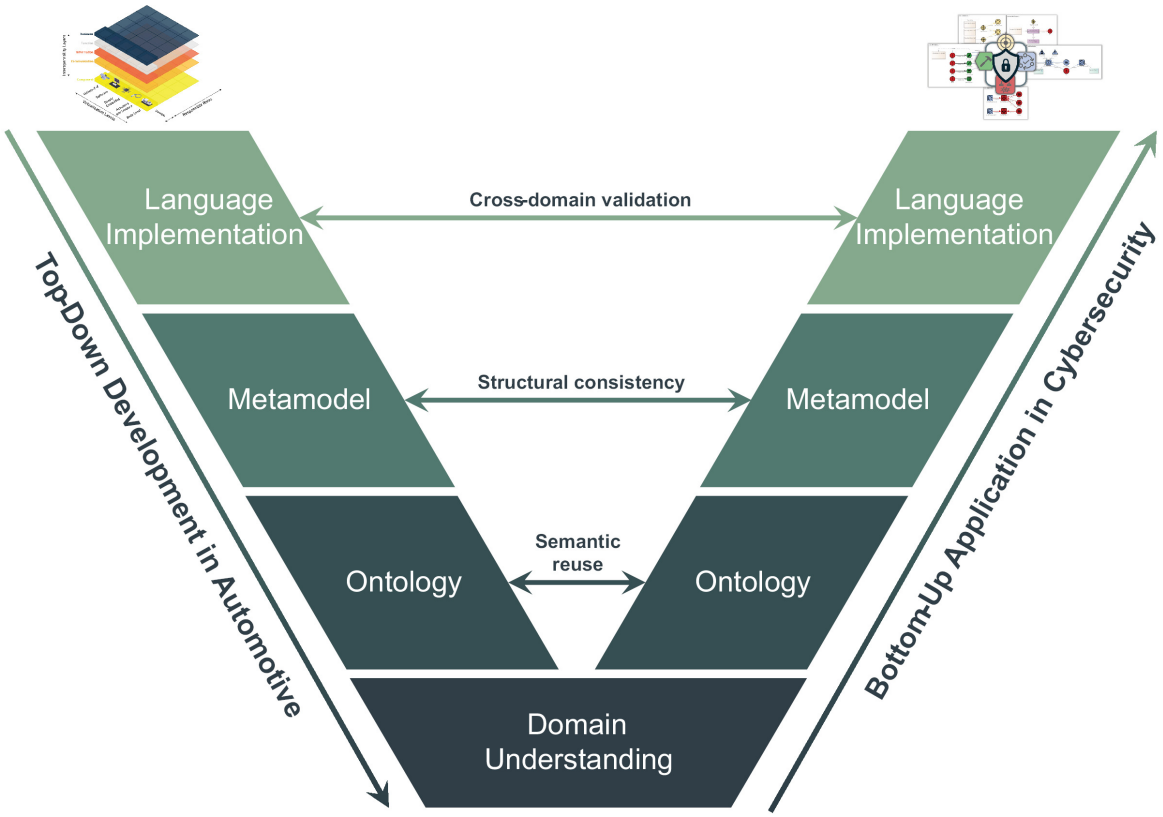

Figure 2. Development and application of the language specification process, inspired by the V-model

The right side of the V represents the bottom-up application of the process in the cybersecurity domain. This bottom-up approach not only validates the applicability of the defined process but also demonstrates the feasibility for another domain. The application of the process therefore starts with the acquisition of the needed domain knowledge. This knowledge is condensed in a well-structured ontology artifact. Thereafter, the structured semantics and knowledge serve as the basis for the definition of a metamodel. Finally, this metamodel delivers the rules for the implementation of the formally specified DSL.

The horizontal relations within this V-model-inspired approach are semantic and structural consistency, and interoperable language implementations. This horizontal alignment also enables the seamless interoperability between the DSLs from two different domains, as they are based on the same formal specification process.

3.3. Requirements and Design

To answer the defined research questions and ensure the applicability of the proposed process, certain requirements for each artifact and the three prototypes were defined in the respective previous publications and are summarized in the following. First, during the development of the ARAM framework and its DSL, three leading requirements were identified for the DSL implementation. These requirements (Reqx.y) were outlined and discussed by Polanec et al. [12] and are summarized as follows:

- **Req1.1 – Intuitive for domain experts:** The DSL's elements and their visual representation shall be intuitively understandable by automotive domain experts.
- **Req1.2 – Technical detail:** The DSL shall preserve MBSE benefits and technical precision, enabling formal specification, traceability, and well-formed architectural models.
- **Req1.3 – Scalability for complex systems:** The DSL shall scale from small subsystems to complex vehicular systems within the automotive domain.

 doi:10.20944/preprints202605.0788.v1

Second, to ensure that the metamodel serves as a solid foundation for the DSL implementation, Polanec et al. [21] defined the following requirements for the metamodel artifact:

- **Req2.1 – Unified artifact:** The abstract syntax, semantics, and concrete syntax of the DSL shall be integrated into a single metamodel artifact to prevent inconsistencies.
- **Req2.2 – Tool independence:** The metamodel shall be tool-independent, adhering to the MOF standard and implementable in any UML-supporting tool.
- **Req2.3 – Human readability:** The CSM-approach shall be human-readable and accessible for users.
- **Req2.4 – Basis for possible automation:** The metamodel—especially the CSM—shall be defined in a way that enables future automation of the transformation into a UML profile.

Third, for the ontology artifact we originally did not define explicitly formulated requirements, however, the following aspects are implicitly part of our previous publication [22]:

- **Req3.1 – Tool-independent formalism:** The ontology shall be specified in a standard, tool-agnostic format.
- **Req3.2 – Mappable to MOF:** Ontology constructs shall be systematically mappable to MOF/UML elements, enabling automatic metamodel derivation.
- **Req3.3 – Cross-domain applicability:** The ontology approach shall generalize across domains (demonstrated with the ARAM automotive case and intended for cybersecurity) supporting reuse.

These main artifacts are three consecutive steps building upon each other. Therefore, it is essential that they are semantically and structurally consistent. To verify this consistency, three prototypes were developed. For all prototypes the core functionality requires the successful transformation from one artifact to another. They fulfill the following design goals (DGx) [24]:

- **DG1 – Lightweight implementation:** To support future integrability in other tools and to allow automated execution via CI/CD pipelines, all prototypes are implemented as standalone command-line tools.
- **DG2 – Compatibility:** The output generated by ONTOMO is directly compatible with the input format required by MOPRO as well as MQSys.

Based on our previous research, this work initially introduces the overall concept and interdependencies between all methods, artifacts and prototypes, resulting in a unified language specification process and toolchain. The subsequent section outlines the development journey of the three methods that later constitute this process.

4. Development of Artifacts

As outlined on the left side of the the V-model approach shown in Figure 2, the three artifacts—implementation, metamodel, and ontology—were developed and demonstrated using the ARAM DSL. Their development followed established standards such as MOF, UML, RDF, and OWL, and built on best practices and processes from literature, resulting in three methods. The following subsections describe how each method was derived from the existing knowledge base and how this knowledge was extended through the experience gained during their development. In line with the DSR paradigm, this section corresponds to the rigor cycle (Figure 1), linking knowledge base and design.

4.1. Developing the ARAM DSL Implementation

As outlined in our previous work [12], the ARAM framework was designed to address the challenges of modeling complex automotive systems in a manner that is both accessible to domain experts and sufficiently detailed for technical analysis. A key step in this context was the development of a DSL that makes the framework's concepts applicable in real-world scenarios. For every interoperability layer of the framework, the DSL defines a set of elements that support the structured development of an automotive system according to the purpose of the respective layer. Its content therefore comprises

 doi:10.20944/preprints202605.0788.v1

modeling elements such as business actors, functional components, data objects, and communication protocols, as well as the relationships between them.

The DSL was implemented in EA using the MDG technology. This mechanism not only facilitates the integration of the DSL into the modeling tool but also enables the definition of so-called shapescripts to customize the visual appearance of elements. As our research focuses on graphical DSLs, this capability for visual customization is considered a key feature of the chosen modeling environment.

Following the MDG technology, a UML profile was defined in which each ARAM element is realized as a stereotype extending a native UML metaclass. This allows the elements to be utilized within EA through drag-and-drop modeling, while maintaining compatibility with the tool's built-in mechanisms. For example, as shown in Figure 3, the *ARAM::Functional Component* element extends the *UML::Component* metaclass and is customized with a distinct shape and notation. The *ARAM::Technical Component* and the relationship *ARAM::technical realization*, on the other hand, are realized as *UML::Component* and *UML::Realization* relationship respectively but left visually uncustomized. Additionally, between *ARAM::Functional Component* and *ARAM::Technical Component* the metarelationship `metaclass = ARAM::technical realization` is defined. In combination with the two metaconstraints `umlRole=target` and `umlRole=source`, these connectors indicate that an *ARAM::Technical Component* can be connected to an *ARAM::Functional Component* via an *ARAM::technical realization*, whereas the opposite direction is not valid.

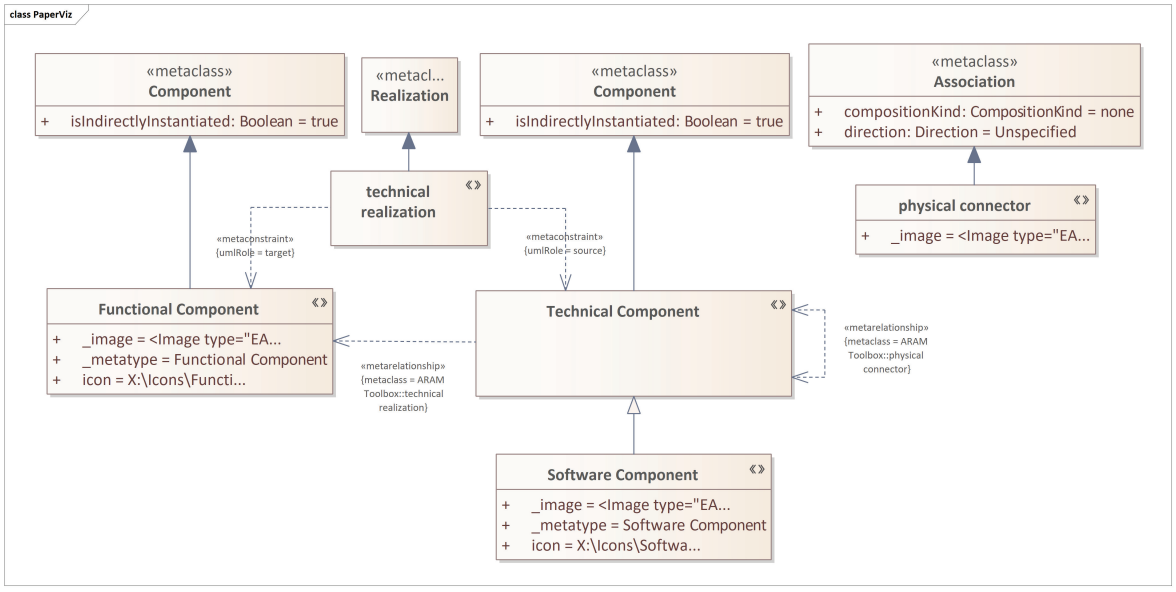

Figure 3. Excerpt of the MDG definition for the ARAM DSL in Enterprise Architect

Figure 4 shows two exemplary shapescripts together with their resulting visualization in EA. The script of the *ARAM::Functional Component* defines the representation of a *UML::Component* (`drawparentshape()`) and a specific fill color (`setfillcolor(255, 255, 0)`). In contrast, the *ARAM::Software Component* element is represented by a custom image (`Image("Software",0,0,100,100)`).

 doi:10.20944/preprints202605.0788.v1

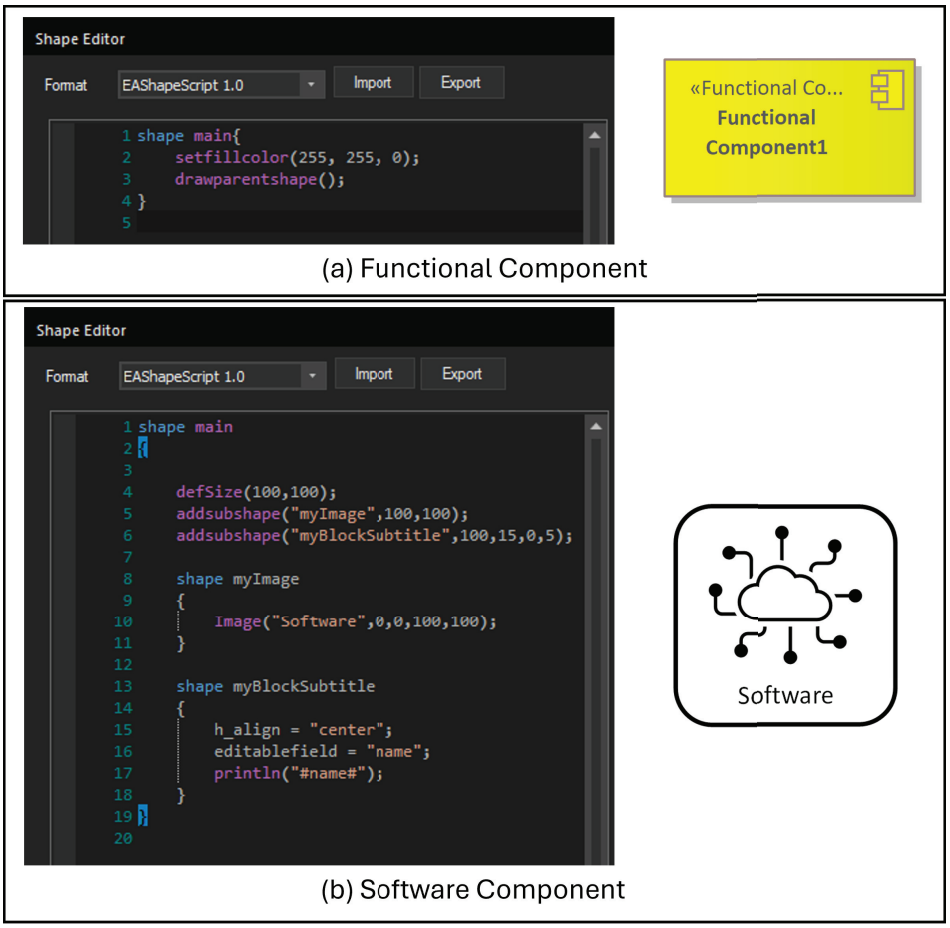

Figure 4. Excerpt of the MDG definition for the ARAM DSL in Enterprise Architect

The design of the DSL was guided by a set of requirements that were already introduced in Section 3. These requirements were addressed as follows [12]:

- **Fulfillment of Req1.1 – Intuitive for domain experts**: The DSL elements were directly derived from the ARAM framework and refined in collaboration with industry partners. The generated illustrative representation made the meaning and type identification of elements more intuitive for domain experts on first sight of generated diagrams. The element's names were derived from terminology commonly used in the automotive domain to avoid an unnecessary steep learning curve to familiarize with generic modeling concepts.
- **Fulfillment of Req1.2 – Technical detail**: By implementing the DSL as a UML profile, the language preserves the rigor and precision of UML-based modeling, while still providing customized notations and stereotypes. This ensures that stakeholders can model with an accessible vocabulary without sacrificing the benefits of MBSE.
- **Fulfillment of Req1.3 – Scalability for complex systems**: The DSL allows modeling at different levels of abstraction. High-level business and functional views can be captured using actors, use cases, and functional groups, while more detailed system views are expressed with technical components, information objects, and communication protocols. This scalability enables consistent use of the same language across multiple abstraction levels.

Altogether, the ARAM DSL implementation provides tool support to apply the framework in practice. While the current realization is tailored to EA, it enables domain experts to create consistent, high-level models of complex vehicular systems and serves as the foundation for the subsequent formalization via metamodel and ontology. This implementation fulfills the third research sub-question (RQ3) by providing an applicable DSL implementation tailored to domain experts through visual customization, including the possibility to integrate company-specific icon sets. At the same time, by

 doi:10.20944/preprints202605.0788.v1

utilizing the standardized UML-profile mechanism, all modeling concepts of UML and MBSE remain supported, ensuring sufficient rigor for detailed technical analyses.

All information about the ARAM framework, including the DSL implementation, is publicly available on our website [3].

4.2. Developing the ARAM Metamodel

While the DSL implementation provides a practical realization as a UML profile, its definition remains tool-specific and hinders effective reusability without a formal language specification. To address this, the underlying structure of the DSL was captured in a metamodel. The metamodel defines the abstract syntax and semantics of the language in a standardized, tool-independent way and thus elevates the DSL from a profile implementation to a formally specified artifact.

The metamodel was defined in accordance with the MOF standard, which is widely adopted for metamodeling in the UML context. Following MOF guidelines, all DSL elements—including the languages relationships—were represented as classes, their properties as attributes, and their interrelations between ASM elements as associations. In addition, semantic information was attached to each element through textual descriptions, forming the SM. Together, the ASM and SM provide a reusable definition of the ARAM DSL's core structure and semantics, as illustrated in Figure 5, using the same exemplary elements as in Figure 3. However, following the guidelines of MOF alone does not capture the graphical representation of the language constructs.

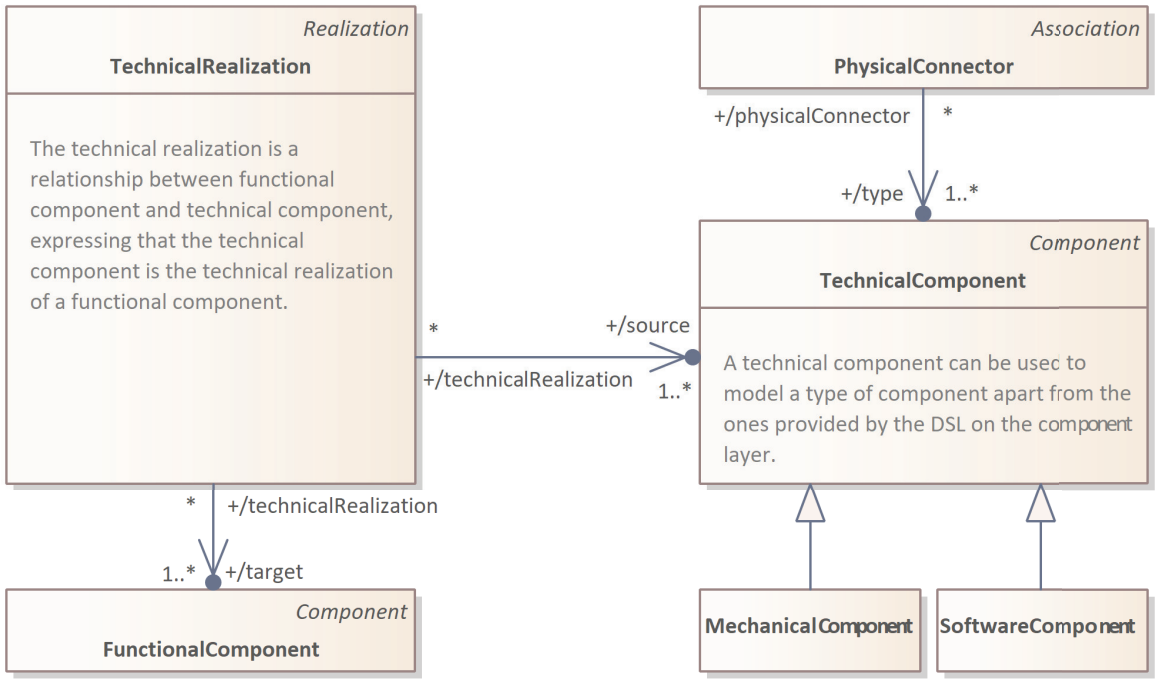

Figure 5. Excerpt of the ARAM metamodel comprising ASM and SM, also shown in [21]

A key contribution of our approach [21] was therefore to extend this metamodel with an explicit CSM. While MOF captures abstract syntax and semantics, it does not define how elements are represented visually, which is an essential aspect for graphical DSLs. To close this gap, we introduced dedicated CSM elements, realized as UML classes, which specify the graphical properties of language constructs. These properties include, for example, shape, fill color, border style, and name alignment for node elements, as well as line style, arrow heads, and direction for relationships. Each ASM element is associated with one or more CSM elements, ensuring a consistent mapping between structure, semantics, and visualization.

[3] www.dsse.at/aram

 doi:10.20944/preprints202605.0788.v1

Figure 6 shows the CSM elements corresponding to the previously described ASM elements. Below the specification, their visual representation in EA is displayed. The *TechnicalComponent* CSM element inherits its attributes from our defined *CSM Entity native* element, since its visual representation is directly inherited from the native *UML::Component*. On the other hand, the CSM element *SoftwareComponent* is realized through a customized image and therefore inherits from the *CSM Entity image*. Finally, the CSM element *PhysicalConnector* is derived from the *CSM Relationship* element, which provides attributes to specify the visual representation of relationship elements. To make the definition of the metamodel more user-friendly the basic representation style of UML parent classes is assumed if not defined otherwise for each attribute.

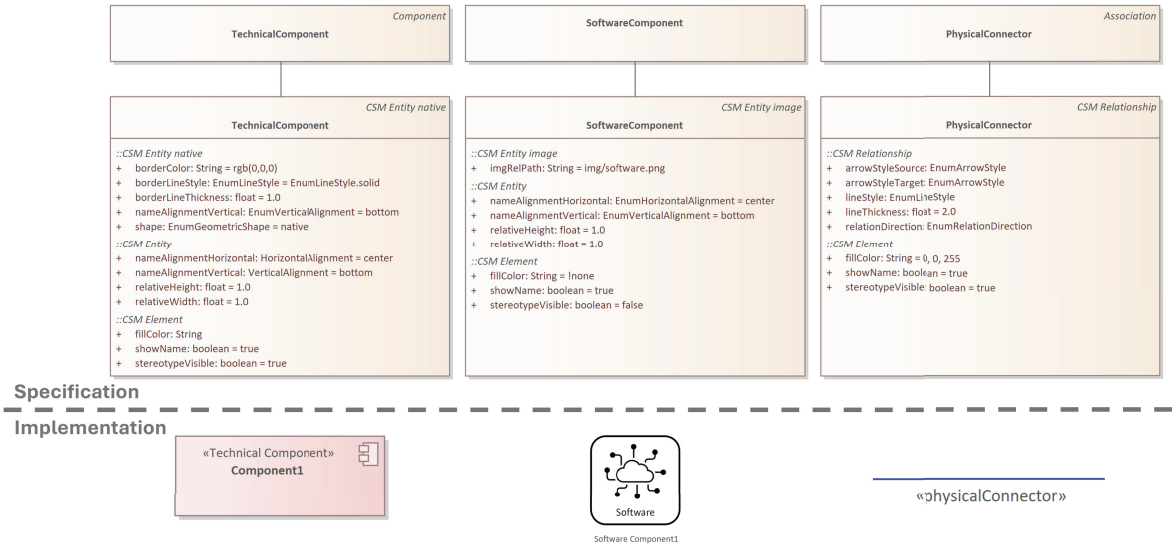

Figure 6. Exemplified representation of the ARAM metamodel including CSM definitions and their implementation, also shown in [21]

The development of this MOF-conformant metamodel, extended by our CSM definition, was guided by the requirements outlined in Section 3. Their fulfillment can be summarized as follows [21]:

- **Fulfillment of Req2.1 – Unified artifact**: By integrating ASM, SM, and CSM in a single artifact, the metamodel avoids inconsistencies between separate definitions and provides a holistic specification of the graphical DSL.
- **Fulfillment of Req2.2 – Tool independence**: Although the DSL is currently implemented in EA, the metamodel itself is modeled using pure UML. It can therefore be exported as an XMI file and imported into any UML-capable tool. Hence, the metamodel artifact is *modeling*-tool independent.
- **Fulfillment of Req2.3 – Human readability**: The metamodel was designed for machine processing but remains accessible to humans through the use of (semi-formal) natural language annotations.
- **Fulfillment of Req2.4 – Basis for possible automation**: This requirement was not fully addressed in this work [21], but demonstrated in a follow-up work [23] by implementing the prototypical transformer MOPRO. The details about this prototype are outlined later in this section.

Altogether, the method of creating the ARAM metamodel artifact contributes a standardized, MOF-based specification of the DSL that unifies abstract syntax, semantics, and concrete syntax. The previously implemented ARAM UML profile served as input for capturing the relevant information of the language, while the resulting metamodel provides a formalized definition that can be reused independently of the original tool-specific implementation. The second research sub-question (RQ2) is thus answered by utilizing and extending the MOF standard, without violating its guidelines. This results in a single metamodel artifact, that also includes the graphical definition of a graphical DSL while remaining fully compliant with the standard.

 doi:10.20944/preprints202605.0788.v1

The complete metamodel, including all ASM, SM, and CSM elements, is publicly available on our website [4].

4.3. Developing the ARAM Ontology

The metamodel provides a formal specification of the ARAM DSL, however, it still relies on a modeling tool and therefore remains primarily tied to model-based artifacts. To provide a tool-independent semantic foundation, we developed a corresponding ontology that captures the core concepts of the DSL, independently of any modeling specifics, in a machine-readable format. This ontology serves as a common vocabulary that can be reused across disciplines, tools, and lifecycle phases, enabling semantic consistency beyond MBSE tools.

The ontology was created following the well established methodology described by Noy et al. [45] which we adapted based on the experience with the ARAM DSL. Figure 7 illustrates the original seven-step process on the left side and our adapted version on the right side.

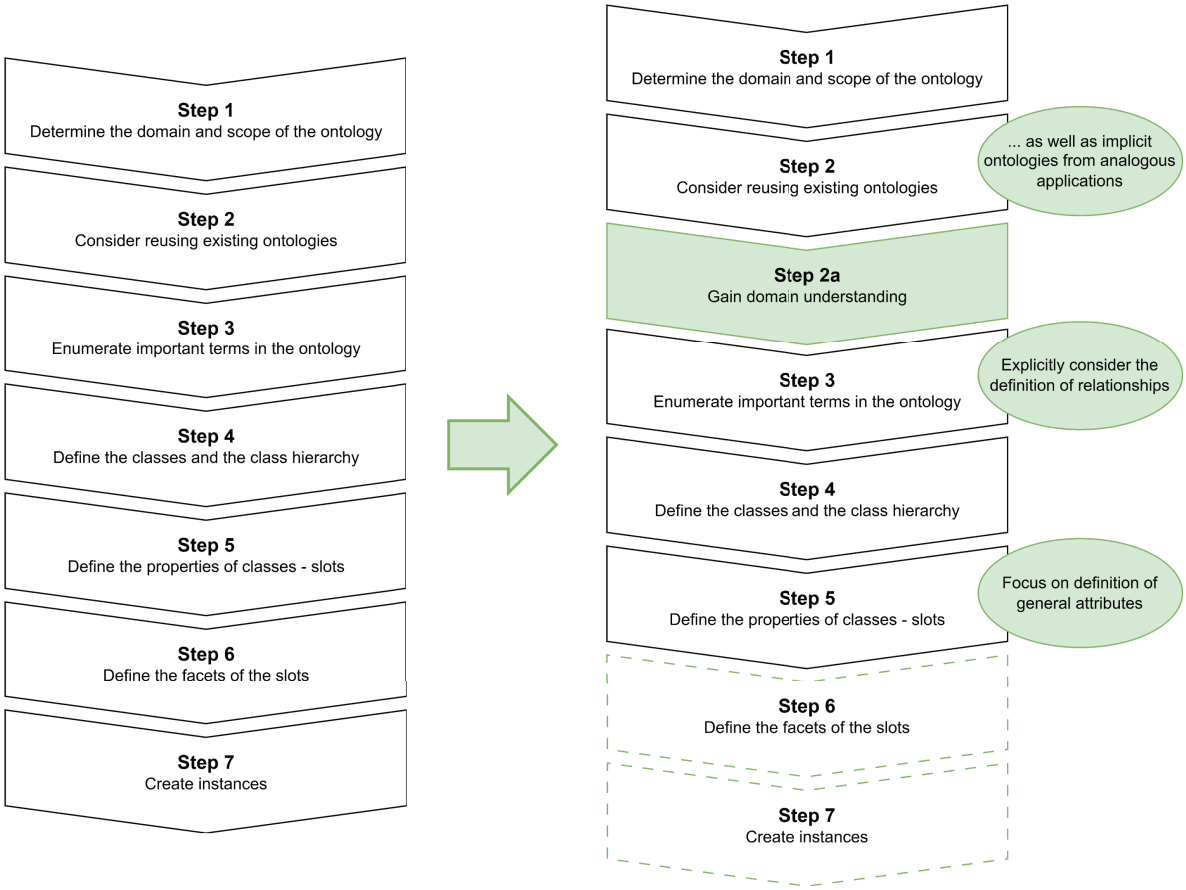

Figure 7. Seven-step process presented by Noy et al. [45] (left) and adapted version (right) as presented in [22]

We started the creation of the ARAM ontology by defining the domain and scope, as proposed by Step 1. This includes the ontology's purpose as the semantic foundation for the ARAM DSL and a set of competency questions [22]. Many of these aspects were already implicitly present in the ARAM framework, metamodel, or profile, but were now made explicit and consolidated.

Step 2 addressed the reuse of existing knowledge. Instead of focusing solely on formally defined ontologies, we extended this step to also consider related DSLs, frameworks, and standards. As ARAM was strongly influenced by SGAM, several concepts were reused or adapted, and communication elements were aligned with automotive standards such as Controller Area Network (CAN), Local Interconnect Network (LIN), and FlexRay.

[4] www.dsse.at/aram

 doi:10.20944/preprints202605.0788.v1

As an additional Step 2a, we explicitly emphasized gaining domain understanding. This includes stakeholder workflows, responsibilities, and terminology, since not all actors in the automotive domain use the same vocabulary. This step is particularly important when developing a DSL that is intended to be intuitive for domain experts.

In Step 3, important terms were enumerated, derived directly from the ARAM DSL's modeling elements and relationships, and structured according to the interoperability layers. Here we adapted the process by treating relationships as class-like concepts, rather than attributes, because of their integral role in DSL specification.

In Step 4 classes were defined based on the previously enumerated terms. Moreover, a class hierarchy, that reflects the five layers of the ARAM framework [12] was defined. Step 5 then focused on defining general attributes of classes. In the original approach, this included also the definition of relationships, as in an ontology they are defined as object properties of the classes. Since we already defined the relationships in Step 3, at this point, only other relevant properties were added. Therefore two additional properties, *Introduced in* and *Featured in*, are added to distinguish where elements originate versus where they can appear within the hierarchy of the framework.

Steps 6 and 7 (defining facets and creating instances) were considered optional in this context, as the ontology's purpose was to formalize the language structure rather than provide instance data or detailed constraints.

Altogether, domain concepts were identified from the framework and DSL implementation and defined as ontology classes. Relationships such as *ARAM::technical realization* or *ARAM::PhysicalConnector* were modeled as object properties and enriched with annotations to capture semantics. The resulting ontology was implemented utilizing OWL and RDF supported by the tool *WebProtégé* [43]. The ontology artifact is publicly available on GitHub [5].

Apart from adapting the ontology creation process, we also compared the ARAM ontology artifact with the ARAM metamodel to clarify their roles in the language specification process. This comparison is summarized in Table 1. The most fundamental benefit of using both artifacts is that the ontology captures formalized domain knowledge and semantics, making it applicable to *any* language implementation. The metamodel then explicitly defines the structure and rules of a *modeling* language and thus forms the subsequent step after the ontology toward the implementation of that modeling language.

Table 1. Comparison of Metamodel and Ontology as shown in [22]

Aspect	Metamodel	Ontology
Purpose	Defines the structure and rules of a modeling language	Formalizes domain knowledge and semantics
Semantics	Implicit, tool- or user-defined	Explicit, machine-interpretable, plain text format
Focus	Focus on modeling elements, such as classes and connectors	Focus on domain concepts and terminology
Tool Dependency	Used within modeling environments, therefore dependent on modeling tools	Higher degree of tool independence due to standardized formats of the semantic web
Reusability	Due to tool dependency limited reuse possibilities	Supports reuse, alignment, and interoperability across domains

The fulfillment of the requirements defined in Section 3 is closely tied to results described in Section 5.1. Their fulfillment can be summarized as follows:

- **Fulfillment of Req3.1 – Tool-independent formalism**: By creating the ontology artifact in WebProtégé using OWL and RDF, a standardized, vendor-neutral, and tool-agnostic format was used thus supporting a tool-independent formalism based on the Ontology 101 guide [45].

[5] https://github.com/K-Polanec/DSL-Specification-Process/settings/access

 doi:10.20944/preprints202605.0788.v1

- **Fulfillment of Req3.2 – Mappable to MOF**: A semantic and structural mapping between ontology and metamodel constructs was developed as later outlined in Section 5.2. Based on this mapping, the ontology elements can directly be translated in MOF/UML constructs.
- **Fulfillment of Req3.3 – Cross-domain applicability**: The ontology development process was developed using the ARAM DSL as an example. Its structure is domain-agnostic built on RDF, OWL, and Simple Knowledge Organization System (SKOS) standards to support reuse. By also applying it to the cybersecurity DSL, as outlined in the following sections, we validated its cross-domain generalizability.

Altogether, the ARAM ontology provides the semantic backbone of the language specification process. It complements the tool-oriented DSL implementation and the formal metamodel by introducing a shared, tool-independent vocabulary. In combination, these three artifacts establish a consistent and reusable foundation for domain-specific language engineering in the automotive domain. The application and adaption of the ontology definition guide from literature [45] together with the comparison between metamodel and ontology, answer the first research sub-question (RQ1) by demonstrating a method of how an ontology can be developed to serve as a semantic foundation for a graphical DSL in a tool-independent manner.

5. Application of Artifacts

In contrast to the rigor cycle described in the previous section, which focused on deriving methods and artifacts from the knowledge base, the following subsections demonstrate the relevance cycle of the DSR paradigm (Figure 1). In this context, the methods originally developed for the automotive domain are applied in the domain of cybersecurity to address practical challenges of model-based security analysis. The application highlights how the resulting ontology provides consistent terminology, how the metamodel structures and visualizes security concepts, and how the implementation operationalizes them within a modeling environment. Altogether, this demonstrates that the methods can be seamlessly transferred and reused beyond the automotive domain, thereby underlining their generalizability and practical value.

5.1. Applying the Ontology Specification to the Cybersecurity DSL

After gaining the necessary domain understanding, the ontology serves as the first formal artifact in developing the cybersecurity DSL. The goal was to provide a consolidated vocabulary that reflects the semantics of established cybersecurity standards. Based on our insights from the automotive domain and the ontology specification process outlined in Section 4.3, the right side of Figure 7 was applied to create the cybersecurity ontology.

As for the ARAM ontology, first the scope of the ontology was defined, here focusing on cybersecurity analysis tasks such as identifying threats, assessing risks, and specifying security requirements.

In Step 2 and 2a, not only existing ontologies were considered for reuse, but also existing domain knowledge in the form of standards. Therefore, concepts from ISO/SAE 21434 [46], the Network and Information Security Directive 2 [47], and ISO/IEC 27001 [48] were considered. These standards provide definitions for core terms such as threat, vulnerability, risk, and countermeasure, which are later also represented as modeling elements in the DSL. Through a mapping-and-merging process, semantic overlaps were harmonized and inconsistent definitions across standards were resolved. This consolidation ensures that cybersecurity terminology can be interpreted consistently, even in cross-company or cross-domain projects.

Thereafter, in Step 3 and 4 the important terms and relationships were enumerated and defined as classes. Terms like *Risk* or *Countermeasure* were defined as ontology classes, whereas relationships like *treatedWith* and *exposedTo* were defined as object properties.

In Step 4 the class hierarchy was defined as well. In the context of the cybersecurity DSL the hierarchy follows different steps in a security analysis process such as *RiskElucidation* or *RiskTreatment*.

 doi:10.20944/preprints202605.0788.v1

Hence, the identified classes were grouped in the appropriate security analysis steps to form a class hierarchy.

Finally, in Step 5 additional properties of the classes were defined. Altogether, this approach results in an ontology artifact that serves as the semantic foundation for the DSL's metamodel and implementation. Just as the ARAM ontology, it was realized in RDF and OWL using WebProtégé [43]. Figure 8 shows a snippet of the cybersecurity ontology in WebProtégé.

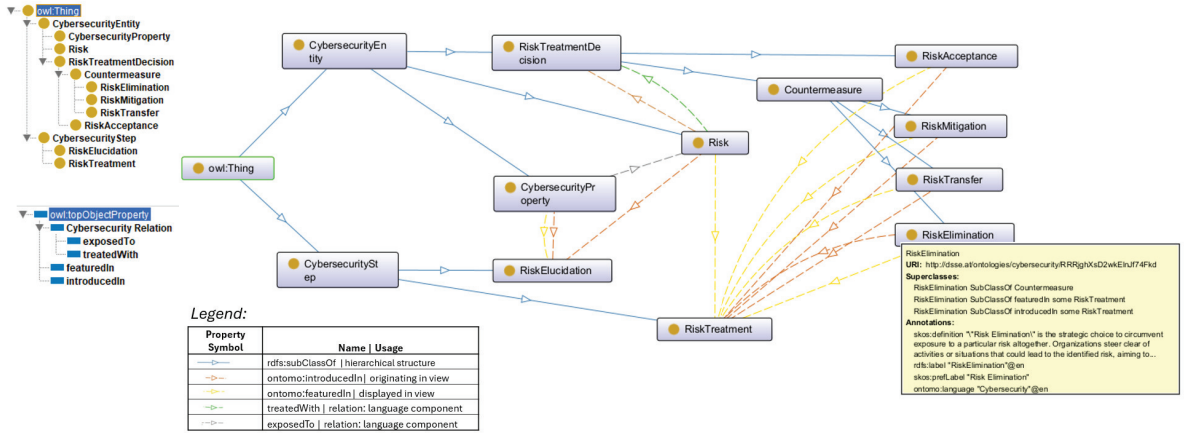

Figure 8. Excerpt of the cybersecurity ontology defined in WebProtégé

5.2. Applying the Metamodel Specification to the Cybersecurity DSL

While the ontology provides the formal semantic foundation of the cybersecurity DSL, the next step is the definition of a metamodel artifact specifying the abstract syntax, semantics, and concrete syntax of the language. Following the same method as for ARAM, the metamodel was realized according to the MOF standard, complemented with our approach for the CSM, thereby ensuring a tool-independent and formally consistent specification.

The previously defined ontology serves as the foundation and therefore as input for the metamodel. For that reason, a mapping between ontology and metamodel concepts was established. Table 2 shows this mapping, originally introduced in [24]. This mapping not only serves as guideline for the manual transformation of ontology to metamodel concepts but also as preparation for automating this transformation process, as later outlined in Section 6.1. Overall, this mapping ensures that the semantics defined at the ontology level are directly preserved in the language specification.

 doi:10.20944/preprints202605.0788.v1

Table 2. OWL-to-UML mapping table, originally introduced in [24].

OWL	UML	Remarks
`owl:Class`	UML Class	Language entity, mapped as standard UML class
`owl:ObjectProperty`	UML Class	Language relationship, also mapped as standard UML class
`rdfs:label`	UML class name	Used as label for both `owl:Class` and `owl:ObjectProperty`
`skos:prefLabel`	Toolbox display name	Used as name to display in modeling environment
`skos:definition`	UML class note	Semantic description rendered as note
`ontomo:language`	Not directly mapped	Indicates belonging to the "Cybersecurity" DSL
`ontomo:languageAbstract`	Abstract UML class	Indicates `owl:Class` or `owl:ObjectProperty` is mapped as abstract
`ontomo:view`	UML Package	Marks a structural language element (with `«view»` stereotype)
`rdfs:domain`	Source element	Source element connected via this relationship
`rdfs:range`	Target element	Target element connected via this relationship
`iri`	GUID input	Used to derive deterministic Enterprise Architect GUIDs
`owl:ObjectProperty = introducedIn`	Mapping instruction	Class is created and displayed in specified diagram (view)
`owl:ObjectProperty = featuredIn`	Reuse indication	Class already exists, only reused and displayed in specified diagram

The metamodel itself was divided into ASM, SM, and CSM. Figure 9 displays a snippet of the cybersecurity metamodel: In the top third of the illustration, the ASM including the SM is shown, which are both modeled following the MOF standard. The SM—which is directly taken from the ontology—is therefore captured in the notes compartments of the UML classes. In the middle third of the figure, the CSM is displayed, which was modeled according to our previously introduced approach [21]. The CSM defines the graphical representation of the language elements, including their shapes, colors, and icons. Finally, in the bottom third of the figure, an example visualization of the elements *Risk* and *Risk Mitigation* as well as the relationship *treated with* are shown, demonstrating how the defined language elements are displayed when utilized in models.

 doi:10.20944/preprints202605.0788.v1

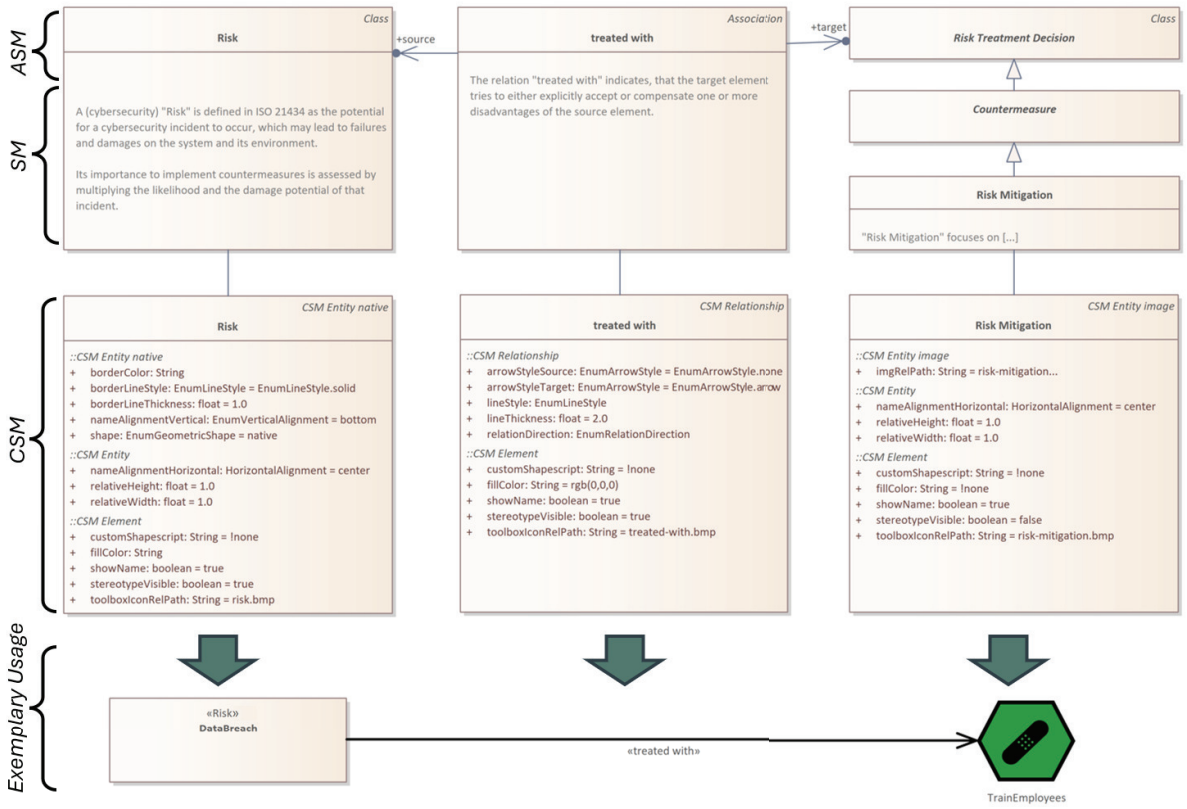

Figure 9. Excerpt of the Cybersecurity metamodel including ASM, SM, CSM, and the final visualization

As the second artifact in the language specification process, the metamodel serves as the formal specification and blueprint of the subsequent language implementation. It includes the semantics already defined in the ontology and extends this general language specification by means of modeling-specific descriptions and rules, also including graphical descriptions. Although the metamodel artifact is modeled in EA, the model itself can be exported as XMI and therefore imported to any XMI-capable modeling tool.

5.3. Applying the Implementation Mechanism to the Cybersecurity DSL

The last artifact in the language specification process is the language implementation. Just like the ARAM DSL, the cybersecurity DSL was implemented as a UML profile in EA using the MDG Technology. Each metamodel element was realized as a UML stereotype extending a native UML metaclass, thereby enabling the modeling of cybersecurity concepts directly within the tool. The language implementation itself is structured like the ARAM DSL implementation outlined in Section 4.1. The resulting MDG Technology is part of the cybersecurity toolbox, which is publicly available[6].

5.4. Application of the Language Specification Process

Following these three steps—first defining the ontology, then the metamodel, and finally the implementation—we established a three-step language specification process. This process, displayed in Figure 10, is based on existing standards and highlights the distinct role and necessity of each artifact within the specification process. The artifacts themselves outline a generalized specification approach that can be applied independently of specific tools. The additional information shown below each artifact in the figure illustrates how we applied this process in practice: the ontology was created in WebProtégé using OWL and RDF, the metamodel was defined as an XMI file in EA, and the implementation was realized as an EA proprietary MDG Technology based on a UML profile.

[6] https://dsse.at/syseng/cybersecurity-toolbox/

 doi:10.20944/preprints202605.0788.v1

Importantly, these tool choices are exemplary rather than prescriptive, and the process itself is not limited to these technologies.

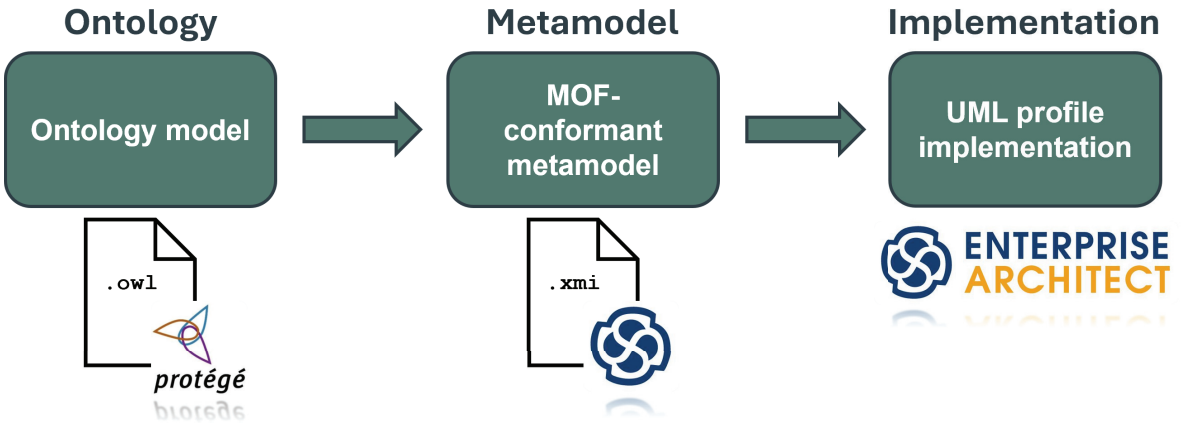

Figure 10. Language Specification Process for graphical DSLs

A central part of our research question concerns semantic consistency between the three created artifacts. While such consistency can be established by manually transitioning one artifact into another, as described in this section, a purely manual approach is error-prone and cannot guarantee correctness, as also pointed out by Hinkelmann et al. [18]. To address this limitation, we developed prototypes that are capable of automatically transforming one artifact into another.

6. Evaluation through Prototypical Automation

The following section completes the DSR approach by outlining the design cycle, as illustrated in Figure 1. As introduced before, the three artifacts were created in consecutive steps by following the proposed specification methods, forming the central specification process as displayed in Figure 10. While this process can be executed manually, we evaluate and verify the semantic consistency and machine readability of the artifacts through prototypical automation. To this end, two prototypes were developed to automate the transitions between ontology, metamodel, and implementation. The subsequent subsections present the development and functionality of these prototypes. Additionally, we introduce a third prototype that demonstrates the adaptability of the toolchain to support other language implementations. Finally, we summarize how these prototypes together enable an automated language specification process.

6.1. Verification of the Consistency between Ontology and Metamodel

To automate the transition from ontology to metamodel, we developed the ONTOMO prototype [24]. Its purpose is to transform an ontology defined in OWL into a MOF-conformant metamodel, thereby ensuring semantic consistency between the two artifacts.

The ontology artifact—which acts as the input for the prototype—contains knowledge about the vocabulary of the language (i.e., elements and relationships), its semantic meaning, and the affiliation of these elements to specific views. This information can be directly translated into the SM and parts of the ASM. However, ontologies by definition do not include certain modeling details. For example, multiplicities and other modeling-relevant attributes are not present and therefore cannot be generated automatically. The same applies to the CSM, as the ontology does not contain information about visual representations of elements. Conversely, the metamodel may contain structural or representational information that extends beyond the knowledge encapsulated by the ontology. Consequently, only those aspects that overlap between ontology and metamodel can be translated automatically. Details

 doi:10.20944/preprints202605.0788.v1

in the ASM, as well as the entire CSM, must be added manually. To support this, ONTOMO generates the skeleton structure of the CSM, so that users only need to complete the visual details.

The actual functionality of ONTOMO is based on a structured mapping between OWL and UML constructs (Table 2). The prototype interprets annotations in the ontology (e.g., `rdfs:label`, `skos:definition`, or custom `ontomo:view`) to preserve vocabulary, semantics, and view affiliation. Using this information, ONTOMO generates the views of the metamodel, creates the corresponding ASM and CSM elements, sorts them into their respective views, and connects them according to the relationships defined in the ontology. To guarantee consistency across updates, each ontology element's `iri` is converted into a Globally Unique Identifier (GUID) compatible with EA. This mechanism ensures traceability between ontology and metamodel and allows incremental updates: if the ontology changes, ONTOMO modifies or reuses existing metamodel elements instead of recreating them.

Technically, ONTOMO is implemented as a lightweight command-line application in C#, following a modular architecture. While the current implementation directly generates the metamodel within the EA environment by interacting with the EA API, the modular design separates the OWL extraction from the export functionality. This allows the prototype to be easily extended to deliver other outputs. Although the prototype produces an EA project as its primary output, the resulting model can be exported as tool-independent XMI, ensuring interoperability with other modeling environments.

To validate its applicability, ONTOMO was applied to two DSLs: the ARAM DSL and the cybersecurity DSL. In both cases, the ontology served as input and was successfully transformed into a corresponding MOF-conformant metamodel, providing the abstract and semantic structure required for subsequent language implementation. Figure 11 exemplifies the transformation of an OWL document from the cybersecurity ontology into a MOF-based metamodel.

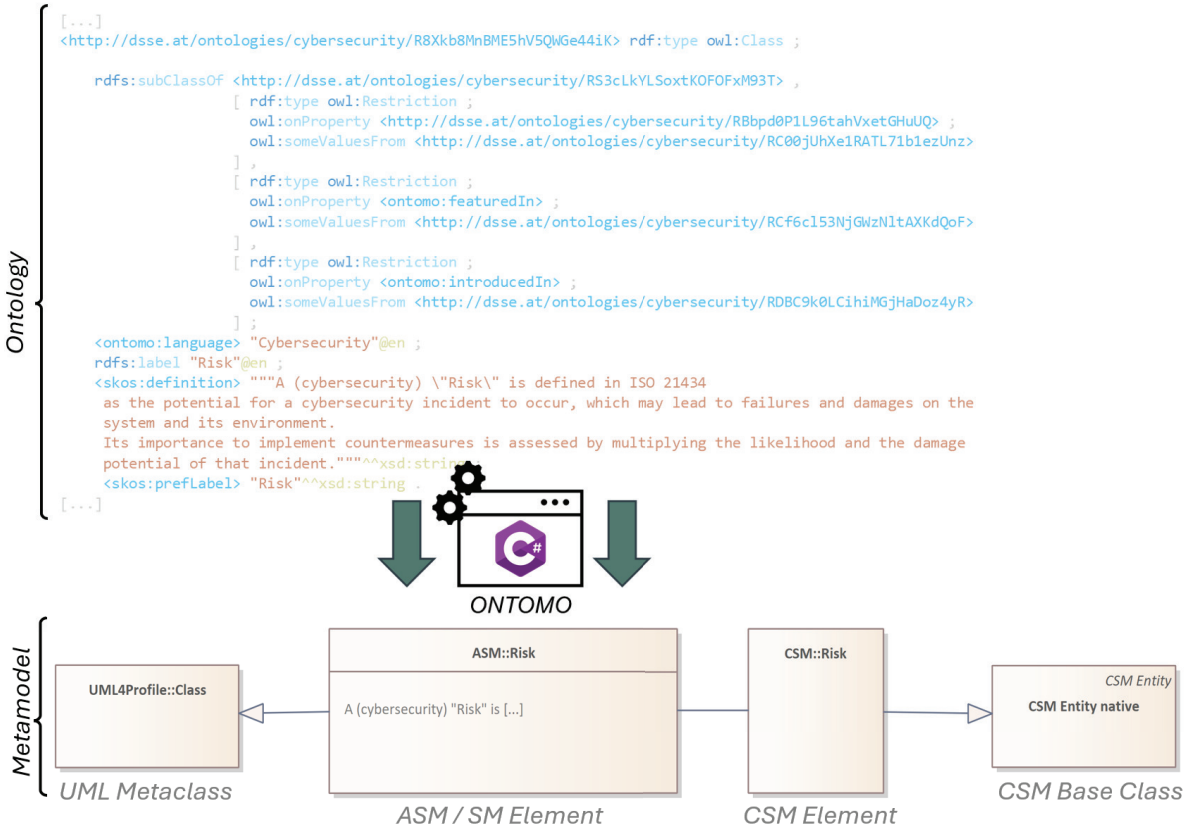

Figure 11. Exemplary transformation of items from the cybersecurity ontology to MOF classes

6.2. Verification of the Consistency between Metamodel and Language Implementation

To further automate the transition from metamodel to implementation, we also developed the MOPRO prototype [22]. Its purpose is to transform a MOF-conformant metamodel into a UML profile

 doi:10.20944/preprints202605.0788.v1

implementation that can be directly applied in modeling tools, thereby providing the final step of the specification process. Together with ONTOMO, which generates metamodels from ontology input, MOPRO completes the automated toolchain, as the output of ONTOMO is directly compatible with the input expected by MOPRO.

A metamodel typically defines the structure of the modeling language, including its abstract syntax, semantic annotations, and the concrete syntax elements required for visualization. While this information is essential for language specification, it is not directly usable within a modeling tool. The role of MOPRO is to bridge this gap by generating the necessary implementation artifacts. Based on the information in the metamodel, MOPRO creates a UML profile containing stereotypes, tagged values, and diagram definitions. In doing so, it translates the metamodel's structural definitions into a format that can be seamlessly integrated into EA.

Specifically, MOPRO parses the metamodel, generates profile elements for each metamodel class, and adds the necessary connectors and tagged values. To support visualization, it also translates concrete syntax definitions into MDG Technology shape scripts, ensuring that the graphical representation defined in the metamodel is preserved in the resulting profile. The generated profile is then imported into EA as MDG Technology, enabling language users to work with the DSL in the same way as with native EA modeling elements.

The prototype is implemented as a standalone command-line application, designed to work directly with the MOF-based metamodels produced in the previous step. Therefore, it takes the EA project containing the metamodel and the file path to the output model as command line parameters. Other than ONTOMO, MOPRO relies on EA for both input and output in its current implementation. However, the transformation process logic itself is generic, as outlined in [23]. Therefore, similar to ONTOMO, also MOPRO was developed with a modular structure, making it adaptable to alternative implementation targets.

To validate its applicability, MOPRO was again applied to both metamodel artifacts: the ARAM as well as cybersecurity metamodel. In both cases, the prototype successfully produced EA-specific MDG Technologies that were imported into the modeling tool. Thereby, MOPRO demonstrated that the formally specified metamodels can be reliably transformed into tool-specific implementations without any additional manual steps, closing the gap between specification and language implementation.

6.3. Verification of the Applicability towards other Language Implementations

The two prototypes that were already introduced in previous research, are both at least partially tool-dependent in their current implementation. However, in both publications we mentioned, that the prototypes are adaptable to support other tools as well. To verify this statement, we developed a third prototype, based on the MOPRO implementation. This third prototype also takes the completed metamodel as input but generates the DSL as a SysML v2 library.

 doi:10.20944/preprints202605.0788.v1

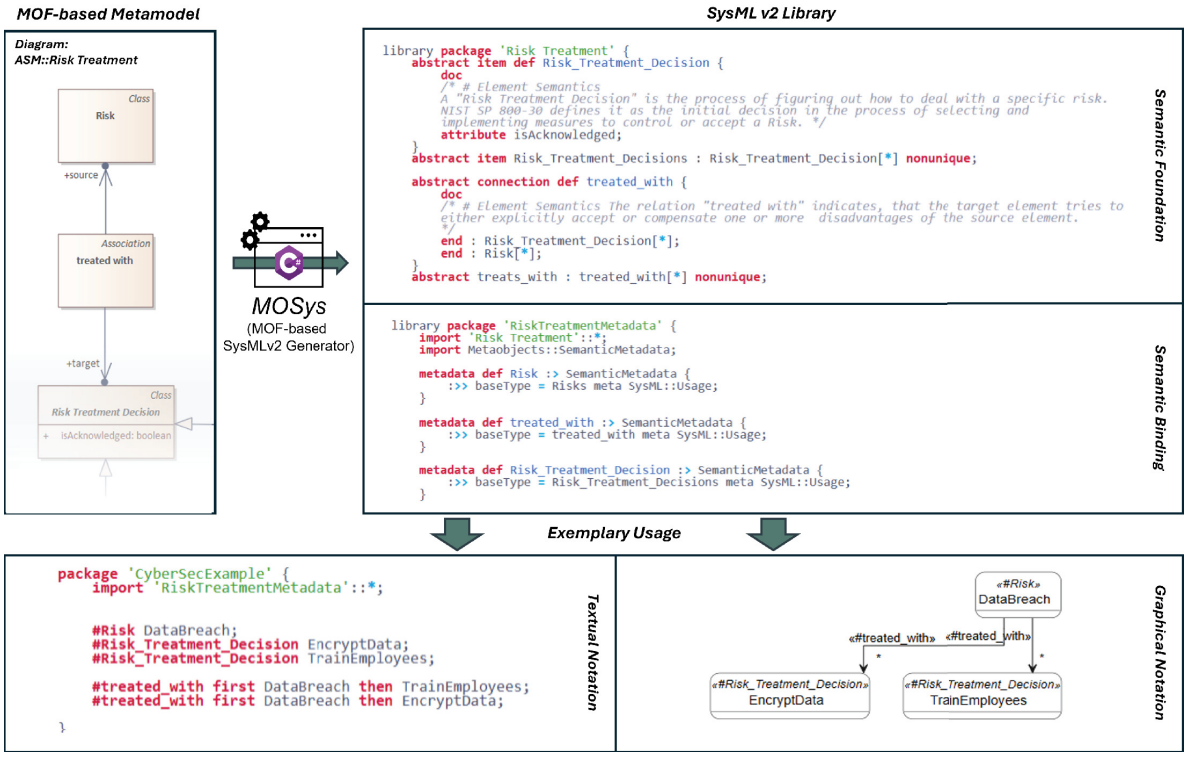

Figure 12. Overview of the MOSys transformation process and an exemplary usage in JupyterLab

Conceptually, MOSys parallels the design of MOPRO. It employs the same command-line interface and metamodel extraction logic, but its output generation logic was redesigned to produce text-based SysML v2 libraries. The transformation process converts each metamodel stereotype into the most semantically equivalent SysML v2 construct (e.g., translating UML class stereotypes into SysML v2 `item` definitions). Through this transformation, MOSys extends the existing ontology-metamodel-implementation toolchain (ONTOMO → MOPRO) with an alternative branch targeting SysML v2 (ONTOMO → MOSys). This demonstrates the conceptual generality of the transformation approach and its applicability beyond EA-specific implementations. The language extension mechanism in SysML v2 is based on the use of SemanticMetadata as defined in the SysML v2 specification [49]. The approach can be understood in two complementary parts:

1. **Semantic Foundation:** Domain concepts are defined using standard SysML v2 constructs, specifying their structural and behavioral characteristics. For example, `def item Risk_Treatment_Decision` defines that it inherits from the SysML v2 `item` construct, lists its properties, and imposes constraints on its use. For connectors, this includes specifying which element types they can link.
2. **Semantic Binding:** A metadata definition creates a domain-specific keyword that binds to the defined concept. This metadata specializes `KerML::SemanticMetadata` and can be applied as an annotation to model elements. When a modeler uses this keyword, the annotated element becomes semantically linked to the concept's full structural definition, inheriting its constraints and semantics rather than being merely a syntactic tag. The `meta` operator establishes this reflective binding.

To validate the approach, we applied MOSys to a subset of the cybersecurity DSL metamodel to generate a SysML v2 library containing both semantic foundations and metadata definitions. This generated library was then imported into a JupyterLab notebook with SysML v2 support, where we created a test model using the domain-specific constructs from the library. Figure 12 illustrates the complete transformation process, from metamodel input through SysML v2 library generation to its application in the JupyterLab modeling environment.

This result demonstrates that the developed toolchain can be adapted to support additional language notations with minimal changes to the transformation logic. Consequently, the approach is

 doi:10.20944/preprints202605.0788.v1

not bound to a single modeling tool or syntax, but rather provides a flexible basis for multi-platform DSL implementation.

6.4. Automation of the Language Specification Process

Through the implementation of the three introduced prototypes, the language specification process illustrated in Figure 10 can be extended by means of automatic transformations into a language specification toolchain as depicted in Figure 13.

The resulting automated language specification process can therefore be described as follows:

1. An ontology is defined following the adapted ontology creation process as introduced in [22].
2. ONTOMO uses this ontology as input and generates the foundation of a MOF-conformant metamodel, as described in [24].
3. The resulting metamodel is then manually completed by adding details to the ASM a visual representation to the CSM, as outlined in [21].
4. MOPRO then takes this completed metamodel as input and automatically generates a language implementation as MDG Technology, which is an extended UML profile for the modeling tool EA. This process is presented in [23].
5. Additionally, a third prototype was developed that is capable of transforming the completed metamodel into a SysML v2 library, therefore demonstrating that the prototypes can easily be adapted to support other output formats

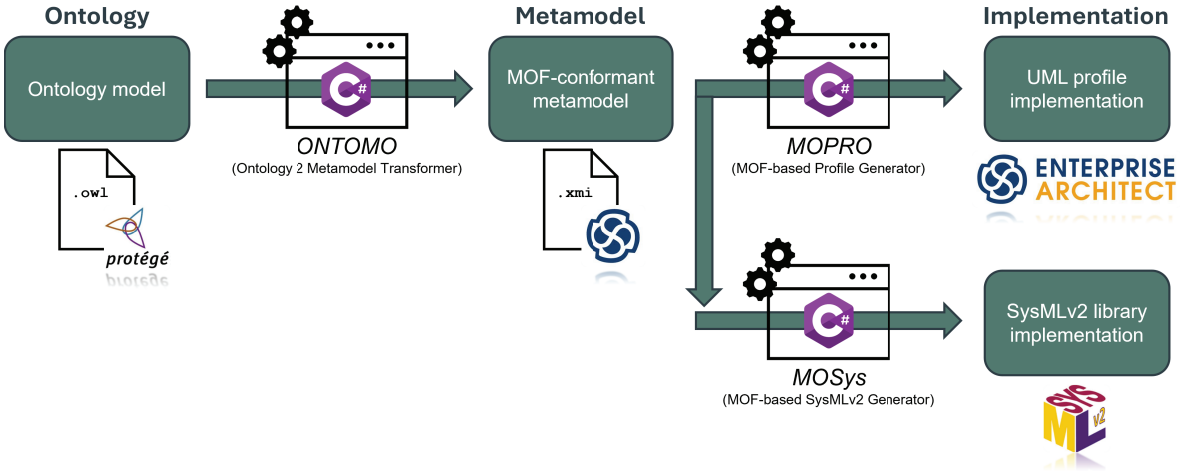

Figure 13. Language Specification Toolchain for graphical DSLs

6.5. Evaluation of the Specification Process

To assess the proposed specification process, we evaluate it along three criteria: semantic consistency, cross-domain applicability, and automation capability. Semantic consistency is addressed through the explicit alignment of ontology, metamodel, and implementation artifacts, as well as through the defined transformations that support the propagation of semantics across these levels (Section 6.1 and 6.2). Cross-domain applicability is demonstrated by applying the process in both the automotive and cybersecurity domains, following complementary top-down and bottom-up approaches (Section 6.3). Finally, automation capability is illustrated by the prototypical toolchain, which enables the automated transformation between artifacts and thereby provides an executable realization of the specification process (Section 6.4). However, it should be noted that the present evaluation is primarily qualitative and based on prototypical demonstrations. A more comprehensive quantitative evaluation, for instance in terms of modeling efficiency, error reduction, or user acceptance, remains subject to future work.

 doi:10.20944/preprints202605.0788.v1

7. Conclusions

In this work, we addressed the lack of a standardized specification approach for graphical DSLs in the context of MBSE. Our guiding research question was how language specifications can be formalized in a way that ensures semantic consistency, reuse, and machine-interpretability across both model-based and non-model-based lifecycle artifacts. We answered this question by introducing a three-step specification process that begins with an ontology as the tool-independent foundation, proceeds to a MOF-conformant metamodel, and finishes in a tool-specific implementation. Each step is grounded in existing standards and best practices, but was extended and adapted according to our research in the context of graphical DSLs.

To support automation and ensure consistency between the artifacts, we developed three prototypes: ONTOMO, which generates a metamodel structure from an ontology; MOPRO, which transforms a metamodel into a UML profile-based implementation; and MOSys, which creates a SysML-v2-library-based implementation. Together, these prototypes demonstrate the feasibility of a standards-based, automated toolchain that bridges ontology, metamodel, and implementation, thereby enabling consistent and reusable DSL engineering.

We validated this process and toolchain using two complementary scenarios: inspired by the V-model, on the left side a top-down approach in the automotive domain, where an existing DSL was stepwise formalized according to the specification process. On the right side of the V-model a bottom-up approach in the cybersecurity domain, where a new DSL was developed from domain knowledge. Both cases confirmed that the proposed approach supports semantic consistency, facilitates automation, and is applicable across domains.

The current work, however, also has limitations. The scope was restricted to graphical DSLs for MBSE, and the implemented prototypes are to a great extend tied to specific modeling environments. By introducing MOSys, we in fact demonstrated, that the adaptability of the prototypes is given, however, ONTOMO and MOPRO are still significantly dependent of EA. Furthermore, the demonstration was limited to two domains.

Future work will therefore focus on three directions. First, we plan to generalize the process beyond MBSE to assess its applicability to other language implementations and non-modeling artifacts. Second, we aim to generalize the implementations of our prototypes, for example by generating XMI-based or Eclipse/EMF-based implementations, to further extend tool independence. Third, we intend to validate the approach in further domains, such as industrial automation or the power systems domain, to demonstrate broader relevance and to assess scalability with larger, more complex DSLs.

Altogether, this work contributes a standards-based methodology and prototypical toolchain for consistent, automated graphical DSL specification. It lays the foundation for further automation and cross-domain reuse, and thus provides a step toward realizing lifecycle-wide semantic consistency in systems and software modeling.

Acknowledgments: The financial support by the Austrian Federal Ministry for Digital and Economic Affairs and the National Foundation for Research, Technology and Development and the Christian Doppler Research Association as well as the Federal State of Salzburg is gratefully acknowledged.

 doi:10.20944/preprints202605.0788.v1

Abbreviations

The following abbreviations are used in this manuscript:

CPS	cyber-physical system
DSL	domain-specific language
MBSE	model-based systems engineering
UML	Unified Modeling Language
SGAM	Smart Grid Architecture Model
RAMI 4.0	Reference Architecture Model Industrie 4.0
ARAM	Automotive Reference Architecture Model
MOF	Meta-Object Facility
INCOSE	International Council on Systems Engineering
SysML	Systems Modeling Language
DSSE	domain-specific systems engineering
OWL	Web Ontology Language
ASM	abstract syntax model
CSM	concrete syntax model
SM	semantic model
XMI	XML Metadata Interchange
OMG	Object Management Group
DSR	design science research
RDF	Resource Description Framework
ONTOMO	Ontology-to-MOF Transformer
MOPRO	MOF-based Profile Generator
EA	Enterprise Architect
MDG	Model Driven Generation
CAN	Controller Area Network
LIN	Local Interconnect Network
GUID	Globally Unique Identifier
MOSys	MOF-based SysML-v2 Generator
SKOS	Simple Knowledge Organization System

References

1. Törngren, M.; Sellgren, U. Complexity challenges in development of cyber-physical systems. In *Principles of modeling: Essays dedicated to Edward A. Lee on the occasion of his 60th birthday*; Springer, 2018; pp. 478–503. https://doi.org/https://doi.org/10.1007/978-3-319-95246-8_27.
2. Orellana, D.; Mandrick, W. The Ontology of Systems Engineering: Towards a Computational Digital Engineering Semantic Framework. *Procedia Computer Science* **2019**, *153*, 268–276. https://doi.org/10.1016/j.procs.2019.05.079.
3. International Organization for Standardization. ISO 15288:2015 Systems engineering - System life cycle processes, 2023.
4. Vogel-Heuser, B.; Böhm, M.; Brodeck, F.; Kugler, K.; Maasen, S.; Pantförder, D.; Zou, M.; Buchholz, J.; Bauer, H.; Brandl, F.; et al. Interdisciplinary engineering of cyber-physical production systems: highlighting the benefits of a combined interdisciplinary modelling approach on the basis of an industrial case. *Design Science* **2020**, *6*, e5. https://doi.org/10.1017/dsj.2020.2.
5. INCOSE. *INCOSE systems engineering handbook*; John Wiley & Sons, 2023.
6. Fowler, M. *Domain-specific languages*; Pearson Education, 2010.
7. Lu, J.; Ma, J.; Zheng, X.; Wang, G.; Li, H.; Kiritsis, D. Design ontology supporting model-based systems engineering formalisms. *IEEE Systems Journal* **2021**, *16*, 5465–5476. https://doi.org/10.1109/JSYST.2021.3106195.
8. Smart Grid Coordination Group. Smart Grid Reference Architecture. Technical report, CEN-CENELEC-ETSI, 2012.
9. Hankel, M.; Rexroth, B. The Reference Architectural Model Industrie 4.0 (RAMI 4.0). *ZVEI* **2015**.

 doi:10.20944/preprints202605.0788.v1

10. Neureiter, C. *A domain-specific, model driven engineering approach for systems engineering in the smart grid*; MBSE4U, 2017.
11. Binder, C.; Neureiter, C.; Lüder, A. Towards a Domain-Specific Approach Enabling Tool-Supported Model-Based Systems Engineering of Complex Industrial Internet-of-Things Applications. *Systems* **2021**, *9*. https://doi.org/10.3390/systems9020021.
12. Polanec, K.; Gross, J.A.; Brankovic, B.; Neureiter, C. Evolution of the automotive reference architecture model towards a domain-specific systems engineering approach. In Proceedings of the 2022 IEEE 27th International Conference on Emerging Technologies and Factory Automation (ETFA). IEEE, 2022, pp. 1–4. https://doi.org/10.1109/ETFA52439.2022.9921592.
13. Kleppe, A. *Software language engineering: creating domain-specific languages using metamodels*; Pearson Education, 2008.
14. Eriksson, O.; Henderson-Sellers, B.; Ågerfalk, P.J. Ontological and linguistic metamodelling revisited: A language use approach. *Information and Software Technology* **2013**, *55*, 2099–2124. https://doi.org/https://doi.org/10.1016/j.infsof.2013.07.008.
15. Walter, T.; Parreiras, F.S.; Staab, S. An ontology-based framework for domain-specific modeling. *Software & Systems Modeling* **2014**, *13*, 83–108. https://doi.org/https://doi.org/10.1007/s10270-012-0249-9.
16. Object Management Group. OMG Meta Object Facility (MOF) Core Specification, Version 2.5.1, 2019. Accessed on: Mar. 12, 2024.
17. International Organization for Standardization. ISO/IEC 21838-1:2021 Information technology - Top-level ontologies (TLO), 2021.
18. Hinkelmann, K.; Laurenzi, E.; Martin, A.; Thönssen, B. Ontology-Based Metamodeling. In *Business Information Systems and Technology 4.0*; Dornberger, R., Ed.; Springer International Publishing, 2018; Vol. 141, pp. 177–194. https://doi.org/10.1007/978-3-319-74322-6_12.
19. Hofferer, P. Achieving Business Process Model Interoperability Using Metamodels and Ontologies. *European Conference on Information Systems* **2007**, pp. 1620–1631.
20. Henderson-Sellers, B. Bridging metamodels and ontologies in software engineering. *Journal of Systems and Software* **2011**, *84*, 301–313. https://doi.org/https://doi.org/10.1016/j.jss.2010.10.025.
21. Polanec, K.; Eschlberger, S.; Gross, J.A.; Millaku, M.; Neureiter, C. A Standardized Specification Approach for Graphical Domain-Specific Languages. In Proceedings of the 2025 IEEE 8th International Conference on Industrial Cyber-Physical Systems (ICPS), 2025. https://doi.org/10.1109/ICPS65515.2025.11087882.
22. Polanec, K.; Eschlberger, S.; Neureiter, C. Demonstrating Ontology-Based DSL Specification in the Automotive Domain. In Proceedings of the 2025 IEEE 30th International Conference on Emerging Technologies and Factory Automation (ETFA), sep 2025. preprint at http://dx.doi.org/10.13140/RG.2.2.15726.93760.
23. Polanec, K.; Riedmann, S.; Eschlberger, S.; Neureiter, C. Introducing a MOF-based Profile Generator to automate DSL Implementation from Formal Metamodels. In Proceedings of the 2025 IEEE 30th International Conference on Emerging Technologies and Factory Automation (ETFA), sep 2025. preprint at http://dx.doi.org/10.13140/RG.2.2.12371.49441.
24. Polanec, K.; Eschlberger, S.; Peter, M.; Neureiter, C. Automating the Bridge between Ontology-Driven Language Engineering and MOF-Based Metamodeling. In Proceedings of the 2025 IEEE 30th International Conference on Emerging Technologies and Factory Automation (ETFA), oct 2025. preprint at http://dx.doi.org/10.13140/RG.2.2.23013.87525.
25. Eschlberger, S.; Vereno, D.; Polanec, K.; Neureiter, C. Introducing the Cybersecurity Toolbox: A Modeling Framework for Smart Grid Security, 2024. preprint at https://doi.org/10.13140/RG.2.2.21743.85925.
26. Haberfellner, R.; De Weck, O.; Fricke, E.; Vössner, S. *Systems engineering: fundamentals and applications*; Springer, 2019.
27. Madni, A.M.; Sievers, M. Model-based systems engineering: Motivation, current status, and research opportunities. *Systems Engineering* **2018**, *21*, 172–190. https://doi.org/https://doi.org/10.1002/sys.21438.
28. Henderson, K.; Salado, A. Value and benefits of model-based systems engineering (MBSE): Evidence from the literature. *Systems Engineering* **2021**, *24*, 51–66. https://doi.org/https://doi.org/10.1002/sys.21566.
29. Object Management Group. OMG Systems Modeling Language (SysML), Version 1.7, 2024.
30. Object Management Group. OMG Systems Modeling Language (SysML), Version 2.0 Beta 4, 2025.
31. Neureiter, C.; Binder, C.; Lastro, G. Review on Domain Specific Systems Engineering. In Proceedings of the 2020 IEEE International Symposium on Systems Engineering (ISSE). IEEE, oct 2020. https://doi.org/https://doi.org/10.1109/ISSE49799.2020.9272214.
32. Object Management Group. Diagram Definition (DD), Version 1.1, 2015.

 doi:10.20944/preprints202605.0788.v1

33. Gupta, R.; Kranz, S.; Regnat, N.; Rumpe, B.; Wortmann, A. Towards a systematic engineering of industrial domain-specific languages. In Proceedings of the 2021 IEEE/ACM 8th International Workshop on Software Engineering Research and Industrial Practice (SER&IP). IEEE, 2021, pp. 49–56. https://doi.org/10.1109/SER-IP52554.2021.00016.
34. Hitzler, P.; Krotzsch, M.; Rudolph, S. *Foundations of semantic web technologies*; Chapman and Hall/CRC, 2009.
35. W3C OWL Working Group. OWL 2 Web Ontology Language Document Overview (Second Edition). Technical report, World Wide Web Consortium (W3C), 2012. W3C Recommendation.
36. Cyganiak, R.; Wood, D.; Lanthaler, M. RDF 1.1 Concepts and Abstract Syntax. Technical report, World Wide Web Consortium (W3C), 2014. W3C Recommendation.
37. Ernadote, D. Ontology-based pattern for system engineering. In Proceedings of the 2017 ACM/IEEE 20th International Conference on Model Driven Engineering Languages and Systems (MODELS). IEEE, 2017, pp. 248–258. https://doi.org/10.1109/MODELS.2017.4.
38. Van Ruijven, L. Ontology for systems engineering as a base for MBSE. In Proceedings of the INCOSE International Symposium. Wiley Online Library, 2015, Vol. 25, pp. 250–265. https://doi.org/https://doi.org/10.1002/j.2334-5837.2015.00061.x.
39. Yang, L.; Cormican, K.; Yu, M. Ontology-based systems engineering: A state-of-the-art review. *Computers in Industry* **2019**, *111*, 148–171. https://doi.org/https://doi.org/10.1016/j.compind.2019.05.003.
40. Wu, S.; Wang, G.; Lu, J.; Hu, Z.; Yan, Y.; Kiritsis, D. Design ontology for cognitive thread supporting traceability management in model-based systems engineering. *Journal of Industrial Information Integration* **2024**, *40*, 100619. https://doi.org/https://doi.org/10.1016/j.jii.2024.100619.
41. Hevner, A.; Chatterjee, S. Design science research in information systems. In *Design research in information systems*; Springer, 2010; pp. 9–22.
42. Object Management Group. OMG Unified Modeling Language (OMG UML), Version 2.5.1, 2017. Accessed on: Mar. 12, 2024.
43. Tudorache, T.; Nyulas, C.; Noy, N.F.; Musen, M.A. WebProtégé: A collaborative ontology editor and knowledge acquisition tool for the web. *Semantic Web* **2013**, *4*, 89–99. https://doi.org/https://doi.org/10.3233/SW-2012-0057.
44. Ma, J.; Wang, G.; Lu, J.; Vangheluwe, H.; Kiritsis, D.; Yan, Y. Systematic literature review of MBSE tool-chains. *Applied Sciences* **2022**, *12*, 3431. https://doi.org/https://doi.org/10.3390/app12073431.
45. Noy, N.F.; McGuinness, D.L.; et al. Ontology development 101: A guide to creating your first ontology, 2001.
46. International Organization for Standardization. Road vehicles – Accessibility – Part 21434: Road vehicles – Accessibility – Vocabulary, 2018.
47. European Parliament and Council of the European Union. NIS 2 Directive, 2022.
48. International Organization for Standardization. ISO27001: Information technology – Security techniques – Information security controls, 2022.
49. Object Management Group. OMG Systems Modeling Language (SysML) Version 20 Beta 2. Technical report, Object Management Group (OMG), 2024.