Towards model-based agent definition with ontologies: A systems engineering approach

Sangeeth saagar Ponnusamy[1,*], Carsten Strobel[1]

[1]Airbus Defence & Space Gmbh, Willy-Messerschmitt-Str 1, Taufkirchen 85521, Germany

Abstract

The adoption of agentic AI within the systems engineering domain necessitates transitioning from fragile, ad-hoc implementations to traceable, verifiable, and industrial-grade software components. To address this challenge, we propose an agent definition ontology that standardizes agent objectives, capabilities, and architectures by formally mapping them across the Requirements, Functional, Logical, and Physical (RFLP) Model Based Systems Engineering (MBSE) framework. By aligning agent specifications with MBSE, this mapping establishes an end-to-end digital thread that bridges high-level requirements with low-level agent implementation. The approach is demonstrated on a simplified drone navigation case study, illustrating how the framework enforces formal interface contracts, maintains architectural traceability, and enables model-based verification for heterogeneous AI agents.

Keywords

Ontologies, Model Based Systems Engineering, Agents, Large Language Model, Reinforcement Learning

1. Introduction

Artificial Intelligence (AI) is increasingly integrated across system design and development to automate engineering workflows and accelerate design cycles. In particular, the deployment of autonomous AI agents—systems capable of independent reasoning and iterative problem-solving via tool invocations—has emerged as a powerful paradigm for managing, simplifying, and improving complex systems engineering workflows. However, current AI agent deployments within engineering pipelines are largely ad-hoc, lacking the rigorous architecture, structural traceability, and data provenance required for safety-critical operations, such as those in cyber-physical systems. This paper bridges this critical gap by proposing the Model-Based Agent Definition (MBAD) framework, which formalizes an explicit agent ontology and anchors it within a model-based systems engineering (MBSE) framework. The remainder of this paper is organized as follows. Section 2 presents the motivation behind this work along with the state of the art. Section 3 outlines the agent definition and MBSE ontologies, detailing the structural mapping rules between them. Section 4 provides feasibility demonstration through an application case study, followed by an outlook on future research and conclusion in Section 5.

2. Motivation & state of art

The application of AI, specifically agentic AI is becoming ubiquitous and is increasingly being viewed as a new paradigm for the design and development of complex cyber physical systems. However, adoption of such agents to system engineering workflows, particularly with stochastic, LLM-based agents introduces critical vulnerabilities such as lack of architectural rigor, traceability, and data provenance to name a few. While frameworks for inter agent communication and with external tools such as Agent2Agent (A2A), Model Context Protocol (MCP) and more recently to standardise knowledge for agents such as

Knowledge Graphs and Model-driven Systems Engineering (KGMDSE), in ISWC 2026 Workshops Joint Proceedings, October 25–26, 2026, Bari, Italy

*Corresponding author.

sangeeth-saagar.ponnusamy@airbus.com (S. s. Ponnusamy); carsten.strobel@airbus.com (C. Strobel)

0009-0001-5389-4437 (S. s. Ponnusamy)

© 2026 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

CEUR Workshop Proceedings ceur-ws.org ISSN 1613-0073

published 2026-11-05

CEUR-WS.org/Vol-4313/kgmdse-paper6.pdf

Open Knowledge Format (OKF) do exist, a systematic approach to definition of agents is still an area to be explored [1].

On the other hand there have been many discussions on ontology driven agents, for instance [2], which focus mainly on exploitation of ontologies by agents, or generation of ontologies by agents such as in [3, 4], however there has not been adequate focus on systematically defining agents using ontologies or any other model based methodologies to the best of our knowledge. Ontologies for agentic AI have been discussed in [1, 5], however these studies tend to focus on building a well defined ontology for agents whereas for practical deployment in industry such ontologies need to be anchored within existing MBSE approaches to ensure consistency and coherency in overall design and development workflows. While there have been discussions on MBSE for Agent Based Modeling (ABM) in general such as in [6], they seldom cover agents driven by AI approaches such as LLM or Reinforcement Learning.

MBSE applies modeling at different stages of development cycle to support requirements, design, analysis, verification and validation by establishing "Single Source of Truth" through its three interconnected pillars of modeling language, tool, and method [7, 8]. In order to implement MBSE, language such as SysML [9] along with methodologies such as RFLP [10], ARCADIA [11], MOFLT [12] supported by tools such as CAMEO etc are commonly employed. These methodologies traditionally segregate problem and solution space thereby enforcing traceability between different layers of abstraction. On the other hand, ontologies which provide formal, machine-interpretable semantics and query capabilities, provide a flexible and complementary approach for SysML based MBSE.

Ontologies for MBSE have been discussed widely in literature [13], for example, Requirements, Functional, Logical, and Physical (RFLP) ontology [10], Structure-Behavior-Function (SBF) ontology [14] and its extended SBFIO variant (incorporating Interfaces and Operating modes) [15] to name few. Specifically, RFLP ontology bridges requirements engineering with multi-tier architectural design, capturing domain-specific concepts alongside standardized foundational ontologies like QUDT [16] for unit and dimensional modeling. RFLP for ABM, specifically, Reinforcement Learning (RL) agents, were discussed in [17] and this paper generalizes it to different class of agents, specifically LLM agents.

In order to ensure consistent development of artifacts downstream, template-based Model-to-Text (M2T) transformation within Model-Driven Engineering (MDE) which generates templates in the target development environment is employed either with classical methods or more recently with AI-augmented methods [18, 19, 20, 21]. However, an approach to generate templates for AI agents particularly exploiting knowledge graphs is found to be lacking and in this paper this will be addressed briefly.

3. Ontology

An Ontology, as defined in [22], provides a framework for organizing domain specific knowledge by defining entities and relationships between them which helps foster standardized communication among stakeholders. When applied to MBSE, ontology provides context and consistency with models accurately depicting elements and their relationships within complex systems [13, 23]. Such a structured representation supports analysis, and reasoning related to system properties, for example, it enables logical deductions and inference with reasoners such as [24], complex querying with SPARQL language [25]. The following section briefly discusses the RFLP system engineering ontology, followed by an agent ontology and the integration between them.

3.1. RFLP ontology

Requirements, Functional, Logical, and Physical (RFLP) ontology establishes a multi-layered taxonomy that bridges abstract engineering intent with physical hardware implementations [10]. Though there exists more detailed variants such MOFLT, R-MOFLT which addresses Mission, Operations etc, in this study, for the sake of brevity it is restricted to RFLP. In essence, this ontology has the following concepts,

- **Requirements Layer (R):** Captures high-level operational intent, stakeholder specifications, and constraints usually expressed in natural text (e.g., *"The system shall trigger a low-altitude alert when the measured altitude drops below 1,000 ft"*).
- **Functional Layer (F):** Defines behavioral logic and state transformations independent of software or hardware implementation (e.g., the function *Compute Low-Altitude Alert*, which continuously evaluates the condition Altitude < 1000 ft to set an alert status).
- **Logical Layer (L):** Models abstract system architectures, software components, and internal data flows without binding to specific implementation (e.g., the logical software component *Flight Warning System* passing an abstract boolean signal *Sig_Alt_Alert* across an internal software interface).
- **Physical Layer (P):** Specifies how the logical layer, L is implemented, usually encapsulates concrete hardware topologies (e.g., the physical *Flight Computer* broadcasting the alert over ARINC-429 Label L, Bit B).

Such concepts have relationships between them and across layers, for example, allocation relations map architectural entities to their behavioral specifications, asserting that a given system performs a designated function. Such relations are formally expressed as a tuple $\langle \text{Logical}, \textit{Performs}, \text{Function} \rangle$ and by Description Logic (DL) axioms [26]. Consequently, automated DL reasoners can derive implicit semantic classifications under the Open-World Assumption (OWA) while constraint validation languages like SHACL [27] can help enforce safety critical requirements, such as multi-allocation across dissimilar physical architectures ($\text{CriticalFunction} \sqsubseteq \exists \text{allocatedTo.Sys}_{\text{Primary}} \sqcap \exists \text{allocatedTo.Sys}_{\text{Backup}}$). The ontology, modeled in Protégé tool [28], is illustrated in the Figure 1, though there exists other concepts and relationships, for the sake of clarity they are not shown.

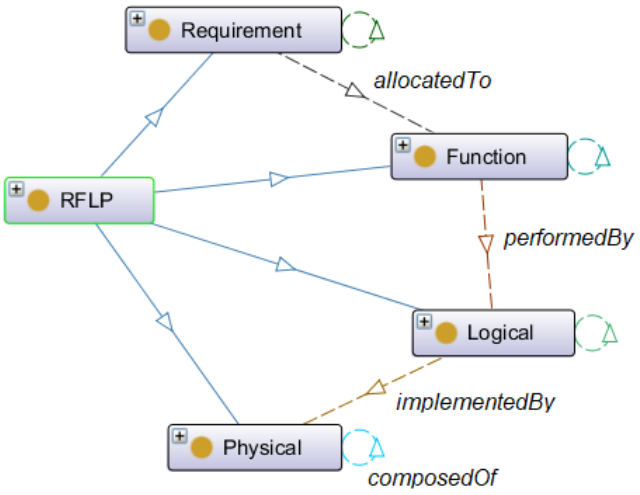


Figure 1: RFLP Ontology

In addition to providing a basis for agent definition in MBSE context, it may be seen from such an ontology that in an agentic workflow, particularly with multi-agents, individual agents could navigate inter-layer trace links ($R \to F \to L \to P$). For instance, a requirement agent operates on R, delegates formal observer compilation to F, maps abstract signal interfaces in L, and grounds final telemetry parameters in P. Such a structured traversal may reduce semantic drift, and improves end-to-end traceability required for human auditing and validation in a MBSE context. The next section discusses a preliminary agent ontology, which leverages standard concepts from LLM and RL agents domain.

3.2. Agent ontology

At its core, an agent is formalized as a goal-oriented entity operating within defined operational boundaries. An agent continuously pursues an explicit objective (*Goal*) managed by an internal reasoning engine (*Reasoning Engine*) and persistent context (*Memory*). It perceives and acts upon its environment through mathematically constrained observation (*Observation Space*) and action (*Action Space*) spaces [29], exchanging typed data payloads (*Trace*) across strictly defined boundary interfaces (*Interface*) and autonomously invoking external solvers or API wrappers (*Tool*). An agent may be deterministic, rule-based *symbolic agents* or probabilistic, data-driven *neural agents* (e.g., LLM planners or RL policies) or may combine both as *neuro-symbolic agents*. Though this paper focuses predominantly on LLM agents and RL agents to an extent, the underlying approach may well be suited to different classes of agents as well. A representative Figure 2 is shown below for illustration where some of the relationships can be seen.

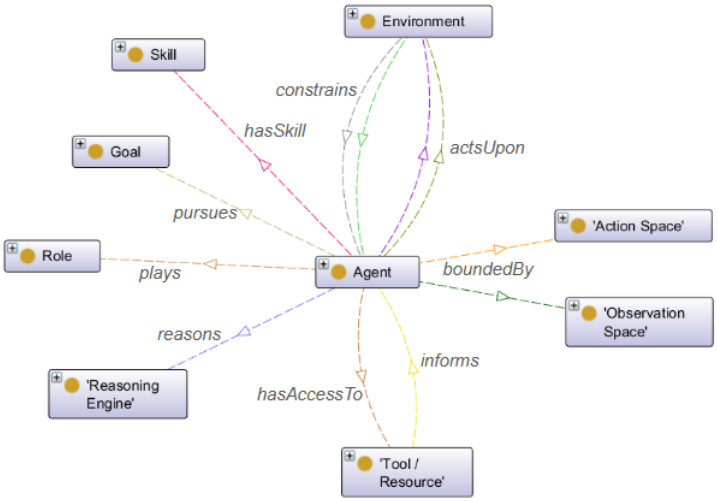


Figure 2: Agent Ontology

Though a one-to-one mapping between different classes of agents may not exist, it is worth pondering similarities that might help in leveraging them to define them in a coherent manner. For example, An agent's *Interface* formalizes its operational boundary by mapping observation space $\mathcal{S}$ to action space $\mathcal{A}$. This manifests as temporal trace with bounds $[\mathbf{x}_{\min}, \mathbf{x}_{\max}] \subset \mathbb{R}^n$ for reinforcement learning agents, and as discrete token trace $\mathcal{V}^k$ governed by structured schemas such as Pydantic for language agents.

3.3. Integration

An overview of the mapping between RFLP and agent ontology is discussed in this section. A representative set of relations between the agent and RFLP ontology is shown in the Table 1. It may be seen that at the requirements layer ($\mathcal{R}$), functional requirements $R_{\text{func}} \in \mathcal{R}$ map via the predicate $definesGoal$ to formal agent objectives $g \in$ *Goal*, while non-functional constraints $R_{\text{non-func}} \in \mathcal{R}$ bound the agent's observation space $\mathcal{S}$ and action space $\mathcal{A}$. At the functional layer ($\mathcal{F}$), abstract system functions $F \in \mathcal{F}$ are allocated to operational capabilities $s \in$ *Skill* executed within cognitive phases $\Phi \in \{\text{Sense}, \text{Decide}, \text{Act}\} \subset$ *ExecutionPhase*, with functional data flows $f_{\text{in}} \in$ *Input* and $f_{\text{out}} \in$ *Output*. These are then allocated to the logical layer ($\mathcal{L}$), where structural components $L \in \mathcal{L}$ map to logical agents $A \in$ *Agent* and its corresponding $I_{\mathcal{L}} \in$ *Interface* typed by T_{schema} or bounded state vectors $[\mathbf{s}_{\min}, \mathbf{s}_{\max}] \subset \mathbb{R}^n$. At the physical layer ($\mathcal{P}$), logical agents are realized by physical software $P \in$ *PhysicalSoftware* instantiating concrete computational engines $E \in$ *ReasoningEngine* such as LLM or RL with typed physical interfaces specifying $interfaceType \in$ {application/json, 'float'...}. Similarly other relations such as *composedOf*, *refines* etc are defined as well among the entities. Such a definition

Table 1
Mapping between RFLP and Agent Ontologies

RFLP	Relation	Agent
Requirement	*definesGoal*	Goal
BehavioralRequirement	*boundsObsSpace*	ObservationSpace
BehavioralRequirement	*boundsActSpace*	ActionSpace
FunctionalElement	*executesSkill*	Skill
FunctionalElement	*decomposesPhase*	ExecutionPhase
FunctionalBehavior	*definesStrategy*	PlanningStrategy
LogicalElement	*allocatedTo*	Agent
LogicalInterface	*contractsInterface*	Interface
Flow	*definesTrace*	Trace
PhysicalSoftware	*realizesEngine*	ReasoningEngine
PhysicalService	*deploysTool*	Tool

establishes a rigorous end-to-end structural traceability, verifiable interface contracts, and integration of heterogeneous AI agents into complex systems.

In the next section, how the constructed knowledge graph could be exploited to build model templates is briefly discussed.

3.4. Template generation

In order to bridge specification and its implementation, in the MDE paradigm, model templates with predefined interfaces are usually generated in the target development environment. In addition to standardised definition, such an approach may guarantee propagation of any interface definition updates in the master model to the generated code artifacts while preserving overwriting user-defined logic. Extending these paradigms to generative AI and multi-agent systems ensures that formal architectural contracts directly govern execution wrappers, such as Pydantic schemas for LLM agents or Gymnasium [29] templates for RL agents, maintaining traceability across layers. An approach to such a template generation has already been discussed in [17] for RL agents in Gym environment, in this paper it has been extended to LLM agents by generating templates in adk environment.

In our ongoing study, using SPARQL queries, system specifications including behavioral personas (*Persona*), skills (*Skill*), memory allocations (*Memory*), tool bindings (*Tool*), and interface contracts (*Interface*) are programmatically extracted from the unified RFLP model. A template engine transforms these SPARQL query results into runtime components: agent templates with Pydantic schemas enforcing structured input/output bounds for LLM agents, Gymnasium *Box* spaces defining observation arrays for RL policies. This model-driven pipeline potentially eliminates manual boilerplates, and helps improving deployment consistency along the development cycle.

4. Example

In order to demonstrate the feasibility of our approach, a simplified drone navigation case study is taken. The drone is assumed to have two levels of navigation modes, namely, strategic and tactical. While the former addresses high level navigation tasks such as mission and path planning, the low level tactical mode is meant for dynamic deconfliction maneuvers and path following. A human controller could guide the drone with natural language commands such as 'Go to zone A, loiter for T minutes and then return to base' which would then be interpreted by the high level LLM agent. This agent which has access to MCP tools such as *get_maps, plan_path* would supply the path to the lower level RL based controller.

In this example, system requirement R_{sys} namely 'autonomous search and recovery in hazardous terrain', is first defined, followed by functional allocation 'tactical search planning' function F_1 which in turn decomposed into low-level 'dynamic control' function F_2. At the logical layer, F_1 is mapped to

a logical 'Mission Planner' L_1, while F_2 is mapped to a logical 'Flight Controller' L_2. At the physical allocation layer, L_1 is allocated to a LLM agent, $A1$ operating over a semantic interface, I_1 defined by a pydantic schema. Conversely, L_2 is allocated to a RL agent, $A2$ operating over a signal interface I_2. Such a mapping could help mitigate the problem of ad-hoc agents integration into formal MBSE artifacts and thereby potentially improving architectural traceability with clearer interface contracts.

Let us take a second example of *Interface* definition and refinement. Initially, the LLM agent, $A1$ processes unstructured natural language operator requests and yields through its interface a validated typed output $A_{\text{O}} = (c, p_{\text{target}}, d_{\text{standoff}})$, where $c \in \mathcal{C}$ denotes the discrete command mode, $p_{\text{target}} \in \mathbb{R}^3$ specifies target spatial coordinates, and $d_{\text{standoff}} \in [d_{\min}, d_{\max}]$ enforces a bounded standoff distance limit. Such an interface definition is refined further to define RL agent $A2$'s signal interface which then maps to target reference setpoints into the RL agent's observation space $\mathcal{S}_{\text{RL}} \subseteq [\mathbf{s}_{\min}, \mathbf{s}_{\max}] \subset \mathbb{R}^n$. The agent $A2$ operating over this bounded state space, its *decide* loop i.e. policy outputs a continuous actuator control vector $\mathbf{a} \in \mathcal{A}_{\text{RL}} \subseteq [\mathbf{a}_{\min}, \mathbf{a}_{\max}] \subset \mathbb{R}^m$, which is also constrained by physical motor voltage bounds $[\mathbf{v}_{\min}, \mathbf{v}_{\max}]$. Such a hierarchical interface refinement establishes better control and traceability across the development cycle of agents of their Interface Control Documents (ICD).

Though the application case is limited in its scope, applying this ontology-based MBSE framework does demonstrate some of the empirical advantages: First, it establishes end-to-end vertical traceability, explicitly linking high-level safety constraints—such as an obstacle separation requirement— to its low-level implementation. Second, querying via SPARQL potentially allows, for example, system architects to run automated impact analyses that could identify which requirements and test cases are affected when swapping or retraining an underlying agent. Third, reasoners perform automated design-time consistency checks by evaluating class and property restrictions; for example, the reasoner can catch an operational boundary conflict where a strategic planner's commanded velocity envelope (v_{target}) violates the restricted property bounds of the low-level RL agent's observation space. Such a semantic integration helps to bridge high-level cognitive planning and low-level dynamic control within a single, more formal and auditable MBSE framework.

5. Conclusion & outlook

A preliminary approach to agent definition via MBSE ontologies is discussed in this paper, and its feasibility is demonstrated with a simplified drone case study. It provides a shared framework for modeling both deterministic, physics-based components and probabilistic, AI-driven elements such as agents at different levels of abstraction. Key challenges include the completeness and validation of this ontology with stakeholders and its integration with existing methodologies and toolchains. Another key challenge is to demonstrate the scalability of our approach without introducing significant additional overhead. One avenue for future work includes leveraging the current SysML v2 paradigm [30] and tailoring it for agent-based specifications. In addition, improving automated configuration template translation tools from agent definitions in the target development environment may help bridge the gap between model-based design and practical AI agent implementation.

Declaration on Generative AI

During the preparation of this work, the author(s) used Gemini-3.5-Flash for grammar and spelling check. After using these tool(s)/service(s), the author(s) reviewed and edited the content as needed and take(s) full responsibility for the publication's content.

References

[1] A. Ekelhart, K. Kurniawan, F. J. Ekaputra, E. Kiesling, AgentO: An Ontology for Modeling Agentic AI Systems, in: The Semantic Web, Springer Nature Switzerland, Cham, 2026, pp. 298–320. doi:https://doi.org/10.1007/978-3-032-25159-6_16.

[2] W.-C. Hung, I.-C. Hsu, Ontology-driven ai agent collaboration for smart healthcare: Integrating model context protocol with mixture of experts, in: Proceedings of the 2025 12th International Conference on Biomedical and Bioinformatics Engineering, ICBBE '25, Association for Computing Machinery, New York, NY, USA, 2026, pp. 149–153. doi:10.1145/3794209.3794236.

[3] J. Recker, F. Hensen, A. Keuper, G. Schuh, LaMAS4PD - A Multi-Agent LLM Approach for Ontology-Driven Structuring of Engineering Knowledge in Industry 4.0, Procedia of the 36th CIRP Design Conference (CIRP Design 2026) 142 (2026) 274–279. doi:10.1016/j.procir.2026.05.260.

[4] A. S. Lippolis, M. J. Saeedizade, R. Keskisärkkä, S. Zuppiroli, M. Ceriani, A. Gangemi, E. Blomqvist, A. G. Nuzzolese, Ontology Generation using Large Language Models, 2025. URL: https://arxiv.org/abs/2503.05388. arXiv:2503.05388.

[5] P. D. Maio, AIAO : AI Agent Ontology v1 (2026). doi:10.6084/m9.figshare.31231783.v3.

[6] P. Acheson, C. Dagli, N. Kilicay-Ergin, Model based systems engineering for system of systems using agent-based modeling, Procedia Computer Science 16 (2013) 11–19. doi:10.1016/j.procs.2013.01.002.

[7] T. Robertson, INCOSE Systems Engineering Handbook, INSIGHT 1 (1998) 20–20. doi:https://doi.org/10.1002/inst.19981220.

[8] J. Estefan, Survey of Model-Based Systems Engineering (MBSE) Methodologies, INCOSE-TD-2007-003-02 25 (2008). URL: http://www.omgsysml.org/MBSE_Methodology_Survey_RevB.pdf.

[9] OMG, OMG Systems Modeling Language (OMG SysML), Version 1.3, 2012. URL: http://www.omg.org/spec/SysML/1.3/.

[10] S. Kleiner, C. Kramer, Model Based Design with Systems Engineering Based on RFLP Using V6, in: M. Abramovici, R. Stark (Eds.), Smart Product Engineering, Springer Berlin Heidelberg, Berlin, Heidelberg, 2013, pp. 93–102.

[11] M. Sango, F. Vallée, A.-C. Vié, J.-L. Voirin, X. Leroux, V. Normand, Mbse and mbsa with capella and safety architect tools, in: G. Fanmuy, E. Goubault, D. Krob, F. Stephan (Eds.), Complex Systems Design & Management, Springer International Publishing, Cham, 2017, pp. 239–239.

[12] R. H. Madeira, D. H. de Sousa Pinto, M. Forlingieri, Variability on System Architecture using Airbus MBPLE for MOFLT Framework, INCOSE International Symposium 33 (2023) 601–615. doi:https://doi.org/10.1002/iis2.13041.

[13] L. van Ruijven, Ontology for Systems Engineering, Procedia Computer Science 16 (2013) 383–392. doi:10.1016/j.procs.2013.01.040.

[14] A. K. Goel, S. Rugaber, S. Vattam, Structure, behavior, and function of complex systems: The structure, behavior, and function modeling language, Artificial Intelligence for Engineering Design, Analysis and Manufacturing (2009) 23–35. doi:10.1017/S0890060409000080.

[15] S. Ponnusamy, V. Albert, P. Thebault, A simulation fidelity assessment framework, in: Proceedings of the 4th International Conference on Simulation and Modeling Methodologies, Technologies and Applications - SIMULTECH, INSTICC, SciTePress, 2014, pp. 463–471. doi:10.5220/0005105704630471.

[16] H. Rijgersberg, M. van Assem, J. L. Top, Ontology of units of measure and related concepts, Semantic Web 4 (2013) 3–13. doi:10.3233/SW-2012-0069.

[17] S. Ponnusamy, P. Helle, C. Strobel, Towards an ontological approach for decision making in agent based systems, in: MBSE-AI Integration, Proceedings of the 13th Internationl conference on Model based Software and Systems Engineering, 2025.

[18] X. He, R. Chen, Z. Zhang, Y. Wang, Q. Dong, A hybrid approach for emf code generation:code templates meet large language models (2025). URL: https://arxiv.org/abs/2512.05498. arXiv:2512.05498.

[19] A. Bucaioni, A. Di Salle, L. Iovino, L. Mariani, P. Pelliccione, Continuous conformance of software architectures, in: 2024 IEEE 21st International Conference on Software Architecture (ICSA), 2024, pp. 112–122. doi:10.1109/ICSA59870.2024.00019.

[20] X. He, T. Zan, BIT: A template-based approach to incremental and bidirectional model-to-text transformation, Journal of Systems and Software 216 (2024) 112148. doi:10.1016/j.jss.2024.112148.

[21] A. Bucaioni, A. Di Salle, L. Iovino, I. Malavolta, P. Pelliccione, Reference Architectures Modelling and Compliance Checking, Software and Systems Modeling (SoSyM) 22 (2023) 891–917.
[22] T. R. Gruber, Toward principles for the design of ontologies used for knowledge sharing, International Journal of Human-Computer Studies 43 (1995) 907–928. doi:10.1006/ijhc.1995.1081.
[23] L. Yang, K. Cormican, M. Yu, Ontology-based systems engineering: A state-of-the-art review, Computers in Industry 111 (2019) 148–171. doi:10.1016/j.compind.2019.05.003.
[24] B. Glimm, I. Horrocks, B. Motik, G. Stoilos, Z. Wang, Hermit: An OWL 2 reasoner, Journal of Automated Reasoning 53 (2014). doi:10.1007/s10817-014-9305-1.
[25] SPARQL 1.1 Query Language, Technical Report, W3C, 2013. URL: http://www.w3.org/TR/sparql11-query.
[26] F. Baader, I. Horrocks, U. Sattler, Description logics as ontology languages for the semantic web, in: D. Hutter, W. Stephan (Eds.), Mechanizing Mathematical Reasoning: Essays in Honor of Jorg H. Siekmann on the Occasion of His 60th Birthday, volume 2605 of *Lecture Notes in Artificial Intelligence*, Springer-Verlag, 2005, pp. 228–248.
[27] H. Knublauch, D. Kontokostas, Shapes Constraint Language (SHACL), W3C Recommendation, W3C, 2017. URL: https://www.w3.org/TR/shacl/.
[28] M. A. Musen, The protégé project: a look back and a look forward, AI Matters 1 (2015) 4–12. doi:10.1145/2757001.2757003.
[29] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, W. Zaremba, OpenAI Gym, 2016. URL: https://arxiv.org/abs/1606.01540. arXiv:1606.01540.
[30] M. Bajaj, S. Friedenthal, E. Seidewitz, Systems Modeling Language (SysML v2) Support for Digital Engineering, INSIGHT 25 (2022) 19–24. doi:https://doi.org/10.1002/inst.12367.

6. Online Resources

- A2A
- ADK
- Gemini
- Model Context Protocol
- OKF
- Pydantic
- QUDT