SysDO: A Domain-Level Ontology for Cross-Domain System Design in Aircraft Manufacturing

Zhengyu Liu[1,*], Sina Namaki-Araghi[1], Arkopaul Sarkar[1], Amine Karoui[1], Rebeca Arista[2,3] and Mohamed Hedi Karray[1]

[1]*Université de Technologie Tarbes Occitanie Pyrénées, 65000 Tarbes, France*
[2]*Airbus SAS, 31700 Blagnac, Midi-Pyrénées, France*
[3]*University of Seville, 41092 Seville, Spain*

Abstract

System design in complex manufacturing industries like aeronautics faces challenges of semantic fragmentation across different layers, such as operational, functional, logical, and technical. Existing ontologies lack domain-specific semantics to unify multi-layer interdependencies, leading to interoperability gaps and traceability issues in model-based systems engineering (MBSE). To address this, we propose the System Design Ontology (SysDO), a domain-level ontology extending BFO/IOF-core to formalize cross-layer relationships, ports, functions, and processes. SysDO bridges semantic discontinuities by integrating SysML-aligned hierarchical perspectives (global, factory, hangar, station) and enabling tool-agnostic interoperability. Evaluated through competency questions on an aircraft manufacturing case study, the query results from the knowledge graph under the SysDO schema confirm its utility in retrieving activities, components, and system structures across layers.

Keywords

Ontology, Design model, MBSE, Manufacturing system design, Semantic Web

1. Introduction

System design serves as the backbone of system engineering—a discipline ubiquitous across domains like manufacturing, software, and business [1]. In complex manufacturing industries specifically, the integration of multi-level hierarchies and multi-view perspectives directly impacts efficiency and cost. In aeronautics manufacturing, for instance, fragmented system models lead to semantic discontinuities: operational requirements fail to propagate coherently to technical implementations, cross-departmental collaboration is hindered by the lack of interoperability, and traceability of errors and defaults gets impeded. Current model-based systems engineering (MBSE) approaches, while structured, struggle with heterogeneous tool silos: SysML models for functional design, CAD tools for physical layouts, and simulation suites for discrete event simulation. All of them operate in isolation, forcing manual reconciliations and risking design drift. Even within the system design itself, requirements, functions, operations, and technical specifications remain insufficiently orchestrated.

Existing ontological solutions for Systems Engineering (SE) have offered valuable insights, yet have not fully resolved these gaps. A primary limitation is that they are often case-specific and lack extensibility to more generalized scenarios. Another frequently witnessed limit is that they tend to focus on the specific perspectives, such as one from the operational, functional, logical, and technical views, without adequately integrating the critical relationships and constraints between these views. These approaches cannot retrieve and represent the totality of information of a complex system design model, where relationships and constraints among multiple views are critical for understanding system behavior, cause traceability, and overall architectural integrity.

Proceedings of the Joint Ontology Workshops (JOWO) - Episode XI: The Sicilian Summer under the Etna, co-located with the 15th International Conference on Formal Ontology in Information Systems (FOIS 2025), September 8-9, 2025, Catania, Italy
*Corresponding author.
zhengyu.liu@doctorant.uttop.fr (Z. Liu)
0009-0006-3539-7898 (Z. Liu); 0000-0002-9075-1419 (S. Namaki-Araghi); 0000-0002-8967-7813 (A. Sarkar); 0009-0000-4780-3482 (A. Karoui); 0000-0001-9382-1593 (R. Arista); 0000-0002-9652-5164 (M. H. Karray)

In this research, we develop the System Design Ontology (SysDO) in order to overcome these limitations. SysDO formalizes relationships and interdependencies across Operational, Functional, Logical, and Technical layers to enable tool-agnostic interoperability. The proposed ontology is a domain-level ontology, which is an extension of BFO as a top-level ontology and IOF-core as a mid-level ontology, tested in the frame of the CORAC-DGAC Excelab project[1]. SysDO includes specifications of ports, functions, and processes in the system design. It is evaluated by competency questions that query a representative case of an aircraft manufacturing system design model provided by Airbus.

This work follows LOT4OCES methodology [2], and the structure is as follows: requirements and competency questions [3] are defined in Section 2, related work is reviewed in Section 3. SysDO's classes and properties are presented in Section 4 with a minimum example. It is then evaluated in the context of aerospace manufacturing in Section 5. Finally, limitations and future work are discussed in Section 6.

2. Requirements

To address these fragmentation challenges, we derive requirements focused on integrating certain layers in the MORFLT framework[4][5]. This research focuses on models' heterogeneity and aims to improve their interoperability within aircraft manufacturing systems, following the SysML v1 language[2]. Such production systems are characterized by hierarchical organization and structured interactions across multiple scopes, such as factory level and station levels, as well as different perspectives. The ontology is supposed to support precise semantic representation of entities and relationships in the product and system design, thereby enhancing model interoperability and supporting efficient system integration without dependence on any specific tool or language.

The requirements for our ontology development are based on SysML v1 and placed in the MORFLT framework (Mission – Operational – Requirements – Functional – Logical – Technical). MORFLT is a structured approach in systems engineering deployed in industry in complex systems such as vehicle and aircraft manufacturing. However, for the purpose of this research, we explicitly focus on the Operational, Functional, Logical, and Technical layers (O, F, L, T), omitting Mission and Requirements. These selected layers are recognized among most existing system design approaches. They represent distinct yet interconnected viewpoints in system design, from abstract operational needs to concrete implementation:

The **Operational** layer characterizes the system's intended use and behaviour by analyzing user needs, actors, activities, and scenarios under real-world constraints (e.g., performance, functionality, safety).

The **Functional** layer defines the functions the system must have to fulfil operational needs. Functional need descriptions are considered included in this layer.

The **Logical** layer structures the established functions into a technology-independent system architecture of components and interfaces.

The **Technical** layer translates the logical architecture into deployable implementations, integrating technical choices, physical architecture, and constraints to produce the system design.

The relationships and constraints among these layers are supposed to be interconnected for trade-off, deduction and co-engineering, while in practice, they are often isolated by information silos. To bridge these gaps and enable effective integration across these layers, the ontology must explicitly formalize the interdependencies and traceability between operational, functional, logical, and technical entities and relationships. This ensures that constraints, design decisions, and system behaviors can be traced coherently across layers, addressing the fragmentation inherent in siloed development of models from different perspectives. By establishing these cross-layer relationships, the ontology should act as a unifying framework to support co-engineering, trade-off analysis, and holistic system validation.

[1] https://skyreal.tech/news/skyreal-integrates-the-excelab-project-of-the-dgac/
[2] https://www.omg.org/spec/SysML/

We formulated a set of illustrative competency questions (CQs) from Airbus's trainer aircraft production line as part of requirement engineering for the ontology [2]. A subset of the complete list of CQs is provided as follows to minimally address information from various aspects of system design, for example, structural composition, functional decomposition, material flows between components, and activity sequences:

CQ1: What are the preceding activity and next activity of **trimming** during sheet press forming?

CQ2: Which hangar has the function **manufacturing elementary parts** of wings?

CQ3: What are function parts of the **elementary metallic part manufacturing system**?

CQ4: What type of input and output materials for the subactivities in **surface protection small** in the elementary metallic part manufacturing system?

CQ5: What resources are used for the **final assembly system** of the wing?

CQ6: What are the subactivities of **sheet press forming**? What is the duration of each subactivity?

While these CQs require only the retrieval of explicitly asserted design data, their combination—and, at times, the inference of new facts—is necessary to address higher-level questions that support critical decision-making. For example, a "make or buy" analysis requires aggregating parts-level cost data, production sequences, and alternative material-flow scenarios to compute total production cost. These higher-order questions lie beyond the scope of this paper and are not included in this paper for brevity. Despite that these CQs are written in phrasings preferred by domain experts (e.g., designers, architects), SysDO mostly admits SysML-specific terms as parent classes, which can be extended with various domain-level terms. In section 5, we show how these CQs are reformulated using these domain-level terms.

3. Related work

Ontologies have been widely adopted in diverse domains, including system design, to ensure semantic consistency and interoperability across heterogeneous models and domains. In this section, similar works are presented within the scope of system design and SysML. Other related works are also introduced in this section to support this study.

In the early work before 2015, Wagner et al. formalized the mapping of State Analysis methodology to SysML in [6], providing a structured ontology for capturing and verifying system states, enabling precise architectural constraints and semantics verification. Graves discussed in [7] integrating SysML with OWL, offering a practical approach to representing SysML block diagrams semantically in OWL to maintain design fidelity and facilitate ontology-based analysis.

In the recent decade, a system engineering reference ontology has been developed in [8], extending BFO as a top-level ontology. It focuses on establishing the context and objectives of the digital artifacts being developed in system engineering processes, while integrating their content into a unified and coherent terminology. An ontology-based requirement verification approach is proposed in [9], facilitating automated reasoning and verification in the early design stages of complex systems. This approach addresses design ambiguity and traceability issues, ensuring requirements are clearly formalized and verifiable through ontology reasoning. Lu et al. developed a scenario-based ontology within an MBSE tool chain in [10]. It formalizes simulation implementations and model information for automated integration and verification purposes, demonstrating significant enhancements in simulation processes. Wardhana et al. introduced in [11] an automatic transformation method converting SysML Requirement Diagrams into OWL ontologies, enhancing semantic clarity and facilitating knowledge reuse and analysis across system designs. Zheng et al. proposed in [12] a semantic-driven tradespace framework integrating requirement management, architecture definition, and manufacturing system design through an application ontology, validated in aircraft fuselage assembly case studies. There are other notable works, including [13], focusing on developing ontologies for system and process modeling; however, we could not further discuss them due to the page limit.

Aside from the previously mentioned domain-level and application-level ontologies, top-level ontologies like Basic Formal Ontology (BFO) [14] and mid-level ontologies like Industry Ontology Foundry

(IOF) [15] offer relatively high-level interpretation of general entities but lack domain-specific semantics for system design. Consequently, top-level and mid-level ontologies can provide theoretical foundations and reference frameworks. Still, domain-specific refinement is needed to capture diverse system design information fully in the context of manufacturing.

Additionally, the Ontology Definition Metamodel (ODM) by OMG [16] is a standardized framework facilitating integration between ontology modeling and model-driven architectures (MDA). It aligns ontology languages such as OWL and RDF with UML-based modeling frameworks, ensuring semantic clarity and tool interoperability.

The previously mentioned work advances the understanding of the semantic consistency and interoperability in system design, but also reveals certain limits. One of the limits is that they tend to focus on a single view, e.g., [9] and [11] focus on requirements, [6] focuses on system states. For complex system design, this incomplete handling of multiple interconnected aspects (e.g., technical and behavioral) and inadequate representation of multi-domain or multi-level relationships among them. This issue has been highlighted within the current era of digital transformation as one of the primary scientific challenges in developing relevant services to obtain holistic transparency for decision-makers, where neglecting interoperability among different digital models across an organization could lead to failure in such projects due to the accumulation of digital model silos [17].

4. Ontology Design

We developed the System Design Ontology (SysDO[3]) to represent the entities and relationships models from different perspectives and bridge semantic gaps across manufacturing system layers. SysDO is a Domain-Level Ontology (DLO) built upon Basic Formal Ontology (BFO) [14] as Top-Level Ontology (TLO) and Industry Ontology Foundry Core (IOF-core) [15] as Mid-Level Ontology (MLO). The inheritance relationships among them are shown in Figure 1 and further illustrated below:

- **BFO** is a widely adopted TLO that provides a set of general and abstract classes, properties, and rules from a philosophical point of view. It distinguishes Continuants (entities enduring through time, e.g., physical components) and Occurrents (processual entities, e.g., material flows). BFO provides most of the superclasses and super-properties in IOF-Core, which are inherited indirectly in SysDO. Additionally, some BFO classes are directly inherited as a superclass in SysDO, for example, *bfo: function* and *bfo: site*.

- **IOF-core** extends BFO as MLO to formalize mid-level industrial concepts in multiple domains. It introduces core classes and properties such as *iof-core: planned process*, *iof-core: informational content entity*, and *iof-core: prescribes*, which bridge abstract BFO categories to domain-specific semantics. For instance, IOF-core's *planned process* class refines BFO's *process* to represent a protocol-driven process governed by constraints and schedules and prescribed by some specifications. *Flow* is defined in SysDO as a subclass of *iof-core: planned process*, formalizing material or information transfers in manufacturing workflows.

- We develop **SysDO** as DLO to further extend the concepts of BFO and IOF-core in system design. The classes and properties defined in SysDO focus on scenarios including hierarchical system decomposition, flow constraints, and cross-layer dependencies.

The classes and properties in SysDO are explained further in detail in the following tables: information-related classes in Table 1, function-related classes in Table 2, process-related classes in Table 3 and object properties in Table 4.

In Table 1, we present the SysDO classes that are subclasses of *iof-core: informational content entity*. This *iof-core: informational content entity* is defined as a content or pattern that conveys information about some entity, independent of how it is physically or digitally expressed. Its existence depends

[3]available on GitHub https://github.com/PICS-LGP/SysDO-System-Design-Ontology/tree/main

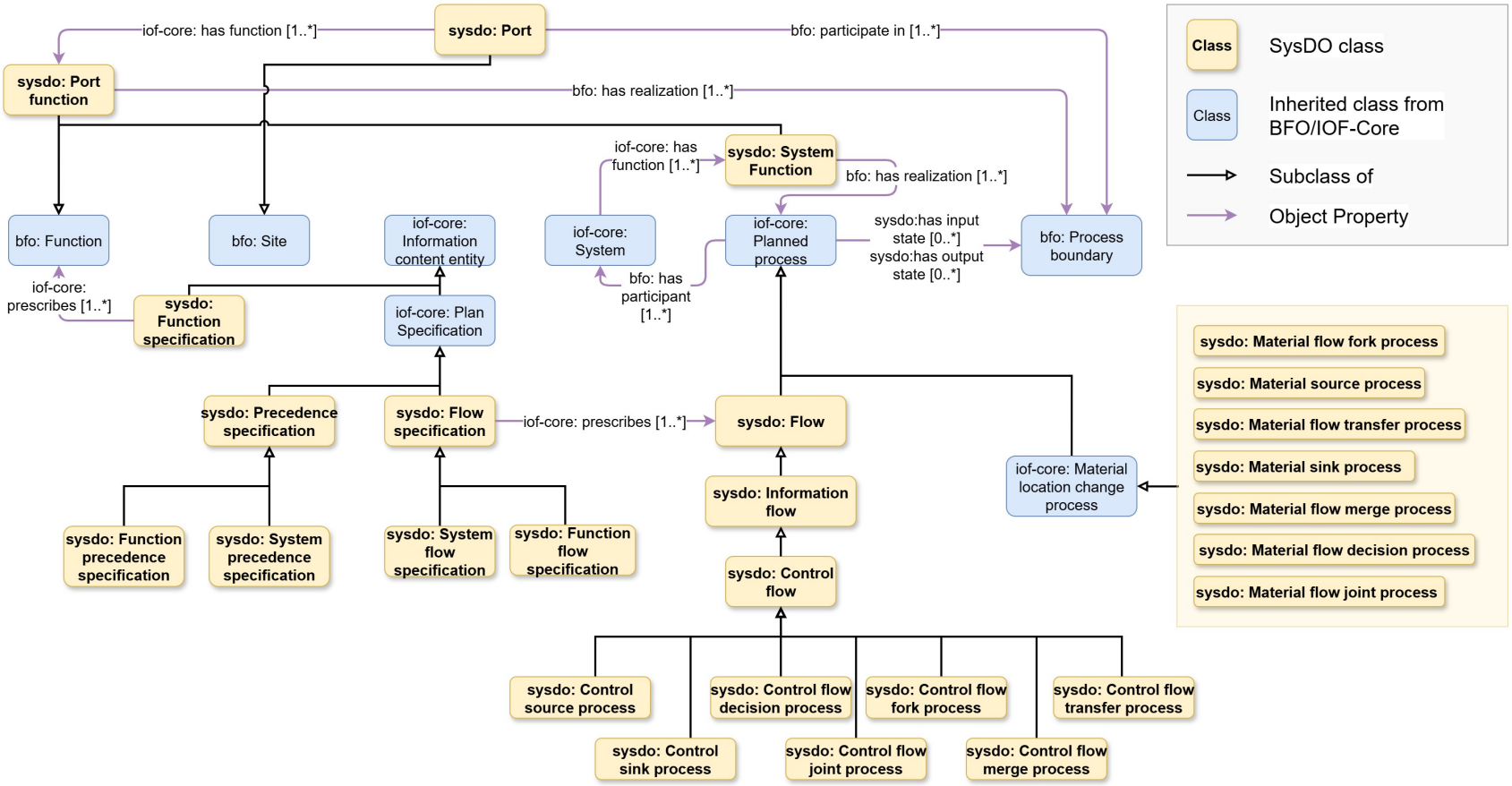

Figure 1: System Design Ontology (SysDO) overview

Table 1
SysDO Classes related to information and specification

Concept	Axiom
`sysdo:function specification`	**Subclass of:** iof-core:InformationContentEntity and (iof-core:prescribes some bfo:function) SubClassOf :FunctionSpecification
`sysdo:receiving function specification`	**Equivalent to** sysdo:FunctionSpecification and (iof-core:prescribes some sysdo: Receiving-Function)
`sysdo:sending function specification`	**Equivalent to** sysdo:PortFunction and (iof-core:prescribedBy only sysdo: SendingFunction-Specification)
`sysdo:flow specification`	**Subclass of:** iof-core:PlanSpecification and (iof-core:prescribes some (sysdo:flow or iof-core:MaterialLocationChangeProcess))
`sysdo:precedence specification`	**Subclass of:** iof-core:PlanSpecification

on physical or digital artifacts (e.g., books, PDF documents), yet its essence is not tied to specific dependencies or carriers. In SysDO, we defined:

- **sysdo:Function specification**: An information content entity that prescribes one or a set of functions to be carried out within a system through the execution of one or multiple processes. For example, the content of a specification document defines the structural joining functions required for aeroplane fuselage assembly. We further defined *receiving function specification* and *sending function specification* for ports presented in Table 2.
- **sysdo:Flow specification**: A plan specification that prescribes transferring materials, information, or energy between functions or components. For example, a logistics plan detailing the movement of the fuselage from the stamping area to the assembly line.
- **sysdo:Precedence specification**: A plan specification that defines the order or dependency between functions. For example, a work instruction stating that the wing must be installed only after it has passed all the quality control phases.

In Table 2, we focus on port and function:

- **sysdo:Port**: A site within a system that serves as an interaction point for data, command, material, or energy exchange. The exchange in ports can be either between processes or entities. For example, a software module with an incoming port for receiving sensor data and an outgoing port for transmitting control signals to an actuator. *sysdo:incoming port* and *sysdo:outgoing port* are defined as subclasses of *sysdo: port* with specified *sysdo:receiving function* and *sending function.*
- **sysdo:Port function** and **sysdo:system function**: These two are subclasses of *function. Function* is a BFO class, defined as a realizable disposition that exists due to an entity's physical make-up, designed to be activated under specific conditions. *bfo:function* is further interpreted in IOF-core, defined as a capability which is the "primary purpose for which the entity was created or evolved". In SysDO, *sysdo:system function* and *sysdo:port function* are derived to represent the intended capability of a system and a port, respectively.

Table 3 focuses on process-related content. SysDO process-related classes are mainly developed under the IOF-Core class *planned process*, which is a subclass of the BFO class *process. Process* is defined as an occurrent that unfolds over time and involves at least one material entity as a participant during some moment of its duration. *Planned process* is a process that follows a specific plan, such as a protocol

Table 2
SysDO Classes related to functions

Concept	Axiom
sysdo:port	**Subclass of:** bfo:site and (bfo:located in at some time some iof-core:System) and (bfo:participates in at some time only bfo:process boundary) and (iof-core:hasFunction only sysdo:PortFunction)
sysdo:incoming Port	**Equivalent to** sysdo:port and (iof-core:hasFunction some sysdo:receiving function)
sysdo:outgoing port	**Equivalent to** sysdo:port and (iof-core:hasFunction some sysdo:sending function)
sysdo:port function	**Equivalent to** bfo:function and (iof-core:functionOf only sysdo:port)
sysdo:receiving function	**Equivalent to** sysdo:PortFunction and (iof-core:prescribedBy only sysdo:receiving function specification)
sysdo:sending function	**Equivalent to** sysdo:PortFunction and (iof-core:prescribedBy only sysdo:sending function specification)
sysdo:system function	**Equivalent to** bfo:function and (iof-core:functionOf only iof-core:System)

Table 3
SysDO Classes related to processes and flows

Concept	Axiom
sysdo:flow	**Subclass of:** iof-core:PlannedProcess
sysdo:information flow	**Subclass of:** sysdo:flow
sysdo:control flow	**Subclass of:** sysdo:information flow
sysdo:control sink process	**Equivalent to** sysdo:control flow and (bfo:precedes max 0 sysdo:control flow)
sysdo:control source process	**Equivalent to** sysdo:control flow and (bfo:preceded by max 0 sysdo:control flow)
sysdo:material sink process	**Subclass of:** iof-core:MaterialLocationChangeProcess and (bfo:precedes max 0 iof-core:MaterialLocationChangeProcess)
sysdo:material source process	**Subclass of:** iof-core:MaterialLocationChangeProcess and (bfo:preceded by max 0 iof-core:MaterialLocationChangeProcess)
sysdo:material transfer process	**Subclass of:** iof-core:MaterialLocationChangeProcess

or instruction. It is characterized by intentional guidance rather than occurring spontaneously or by chance. In SysDO, we have:

- **sysdo:Flow**: A connector representing the transfer of material, information between system components prescribed by a *flow specification*. Flows are classified into material flow and *information flow*, depending on the medium of the flow. Material flow refers to an existing IOF-Core class *material location change process* to avoid redundancy. *Control flow* is defined as a subclass of *information flow* as the control signal is considered a specific type of information.
- **sysdo:Source process** and **sink process**: A source process initiates a flow, while a sink process

terminates it. A source process has no incoming flow and serves as the starting point of a sequence, whereas a sink process receives the final flow, marking the end of the sequence. Depending on the nature of the flow involved, these processes can be categorized as either information (control) or material. A *control source process* triggers the execution of activities, while a *control sink process* deactivates them. Similarly, a *material source process* introduces material into the sequence, and a *material sink process* removes it from the sequence. In SysDO, we also defined sibling classes such as joint, fork, and merge processes. However, they are not discussed here due to the page limit.

Table 4
SysDO Object Properties Related to Continuant and Subsystem Structures

Concept	Axiom
`sysdo:has function part at some time`	**SubProperty of:** has continuant part at some time **Domain:** bfo:Function **Range:** bfo:Function
`sysdo:has process part at some time`	**SubProperty of:** has occurrent part at some time **Domain:** iof-core:Planned process **Range:** iof-core:Planned process
`sysdo:has subsystem at some time`	**SubProperty of:** bfo:has continuant part at some time **Domain:** iof-core:System **Range:** iof-core:System
`sysdo:has input/output state`	**SubProperty of:** bfo:has temporal part **Domain:** iof-core:PlannedProcess **Range:** bfo:ProcessBoundary
`sysdo:has logical subsystem at some time`	**SubProperty of:** bfo:has continuant part at some time **Domain:** iof-core:System **Range:** bfo:site or bfo:object

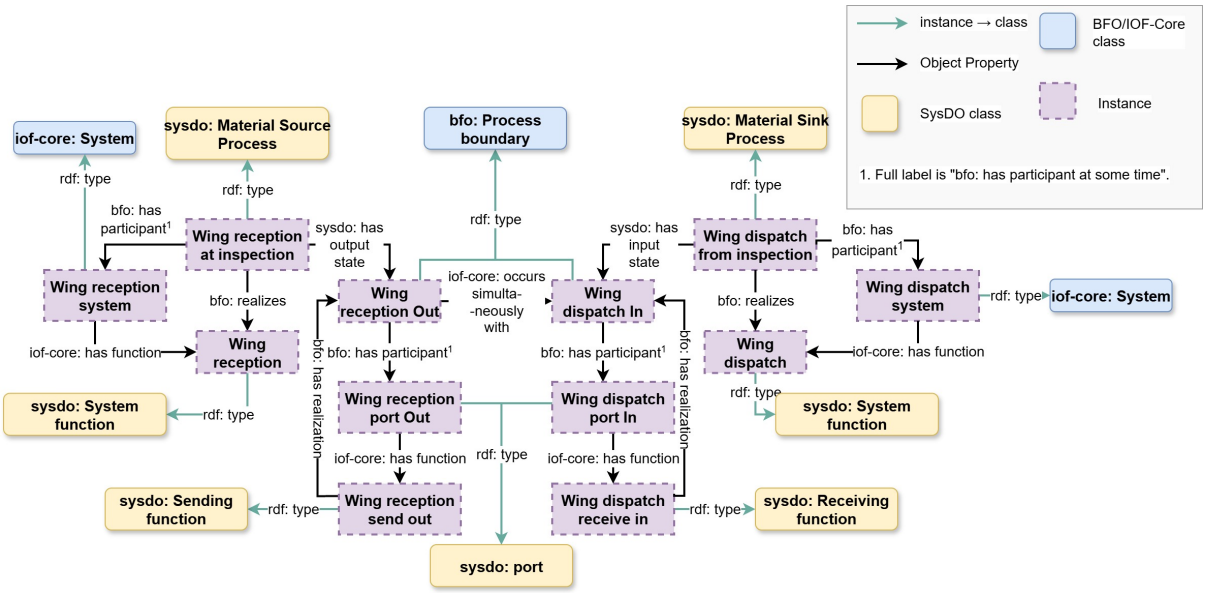

Figure 2: Minimal example illustrating core SysDO ontology concepts

As shown in Table 4, we introduce a set of object properties to model structural and behavioral relations among functions, systems, and processes. Object properties link instances of classes to other instances, unlike data properties, which link instances to literals (e.g., strings, numbers). In SysDO, we have:

- **sysdo:has function part at some time**: relates two functions where one is a functional part that enables the other. As an example, the *function* "body joining" has a function part "spot welding".
- **sysdo:has process part at some time**: relates two processes where one is a subprocess that composes the other. As an example, the *process* "chassis manufacturing" has process part "chassis fabrication" and "quality control".
- **sysdo:has subsystem at some time**: Relates two systems, where one system is a subsystem part of another. For example, in the human body, the digestive system has a subsystem, the stomach, and the circulatory system has a subsystem, the heart.
- **sysdo:has input state** and **sysdo:has output state**: Associates a process with its entry process boundary and exit process boundary, respectively. **Process boundary** is a BFO class; it is defined as a temporal part of a process and the smallest temporal process part without its own temporal part. It is often used to mark instantaneous changes.

In order to better explain how to apply the classes and properties in the SysDO ontology, we present a minimum example in Figure 2 to show how to instantiate ontology classes and properties. In this example, there are two processes in a material flow sequence: a *material source process* (*Wing reception at inspection*) and a *material sink process* (*Wing dispatch from inspection*). In real-world applications, other processes are involved between the source process and the sink process (e.g., static strength tests, fatigue strength tests), with intermediate transformation stages to represent a meaningful quality control sequence. However, this simplified layout was intentionally chosen to provide a more explicit demonstration of core ontological relationships.

- **1. Process system participation**
  - Each process (*Wing reception at inspection*, *Wing dispatch from inspection*) has its corresponding participant system (instances: *Wing reception system*, *Wing dispatch system*).
  - Systems implement system functions (e.g., *Wing reception*), which are realized by their associated processes. For example, *Wing reception* realized by *Wing reception at inspection*.
- **2. Material flow process sequencing**
  - *Wing reception at inspection* (source) transmits material to *Wing dispatch from inspection* (sink) through process boundaries:
    - *Wing reception Out*: Output process boundary of *Wing reception at inspection*
    - *Wing dispatch In*: Input process boundary of *Wing dispatch from inspection*
  - Each boundary involves participant ports (e.g., *Wing reception port Out*, *Wing dispatch port In*), which fulfil specific port functions (e.g., sending function *Wing reception send out*, receiving function *Wing dispatch receive in*) to enable material transfer.

5. Evaluation

In this section, we evaluate the developed ontology by answering the Competency Questions (CQ) posed in Section 2. The evaluation aims to demonstrate the ontology's expressiveness, practical relevance, and its capacity to resolve semantic interoperability and traceability across different MORFLT layers in complex aircraft manufacturing systems.

Specifically, this evaluation is based on a scenario involving an aircraft manufacturing system provided by Airbus, modelled in Cameo[4]. The design model is structured into four hierarchical levels—global, factory, hangar, and station—and analysed through four distinct views: operational, functional, logical and technical. The focus of these views aligns with the four layers (O, F,L, T) from MORFLT introduced

[4]https://www.3ds.com/products/catia/no-magic/cameo-systems-modeler

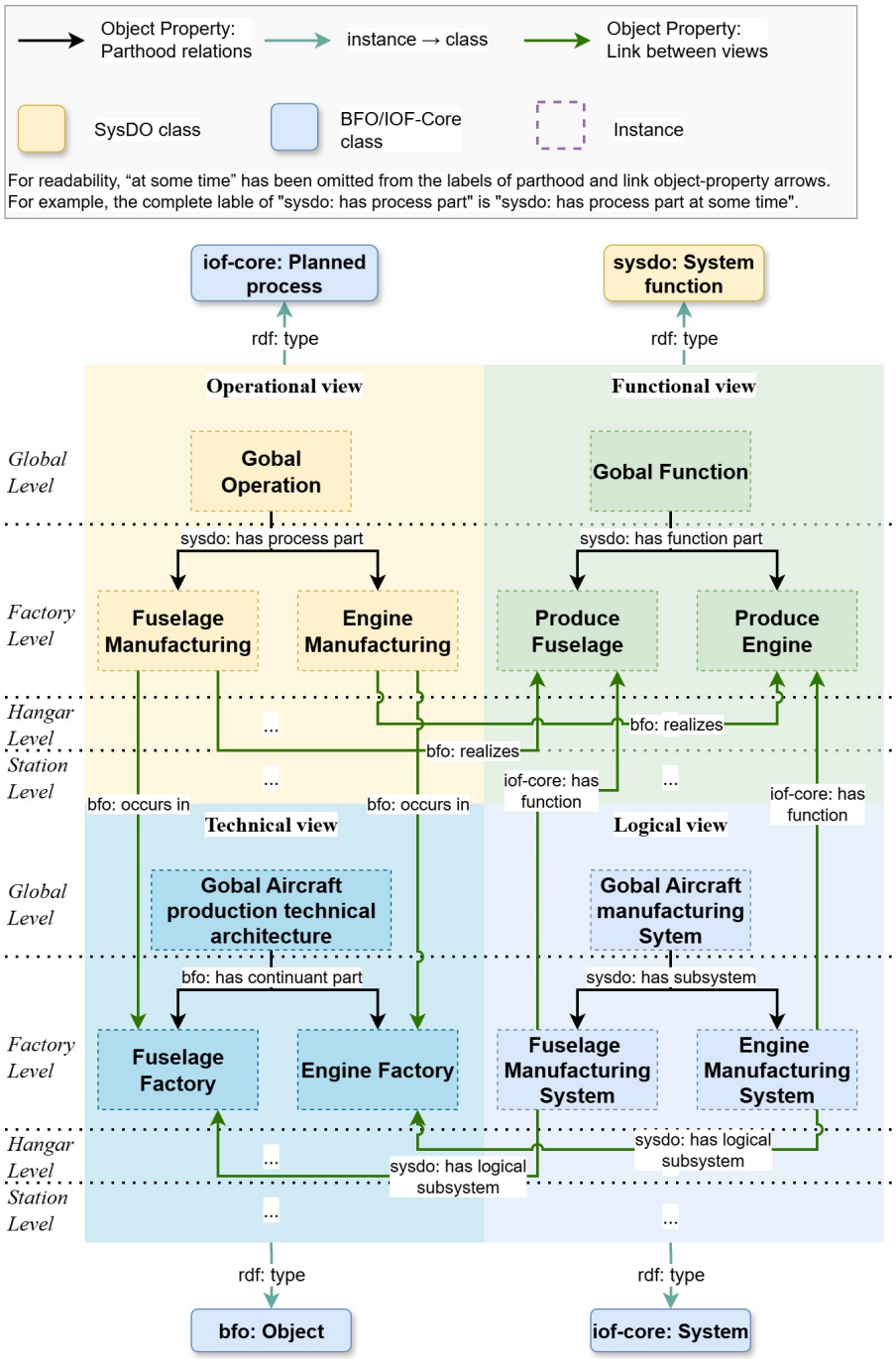

Figure 3: General architecture of aircraft production design model with four views and four levels connected by object properties

in Section 2. Figure 3 presents the general architecture of the aircraft production design model. Note that this diagram is simplified for readability. Some branches and instances, for example, those at hangar level and station level, are omitted in the layout. In each of the four views, there is a corresponding decomposition, e.g., functional decomposition. The parthood relationships are described by a specific object property, based on the ontological class the instance belongs to. For example, in the functional view, an instance is parsed as a “sysdo: system function”, and its corresponding object property linked to the subfunctions is therefore “sysdo: has function part”. Between different views, corresponding instances are linked as follows (the inverse object properties not shown in Figure 3 for brevity):

Operation - Technical are linked by “bfo: occurs in” or reversely by “bfo: environs”.

Operation - Function are linked by “bfo: realizes” or reversely by “bfo: has realization”.

Function - Logical are linked by “iof-core: has function” or reversely by “iof-core: function of”.

Logical - Technical are linked by “sysdo: has logical subsystem” or reversely by “sysdo: is logical subsystem of”.

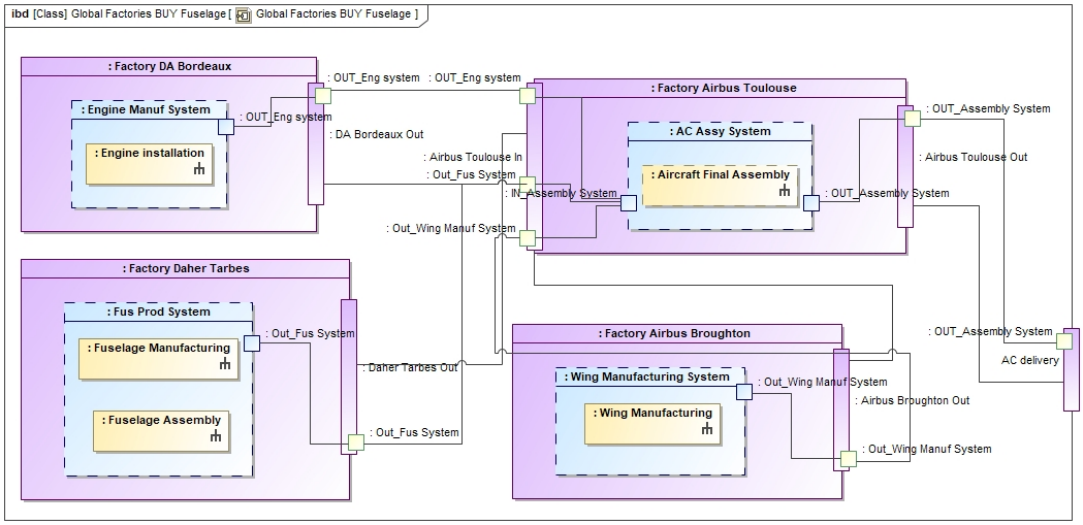

Figure 4: Aircraft global manufacturing system modelled in SysML internal block diagram (IBD)

The model contains multiple diagrams from different views (e.g., decompositions of functions, sequence flows of operations) and various system levels (e.g., factory, hangar). For example, Figure 4 is a SysML internal block diagram that illustrates the technical view of aircraft manufacturing at the global level, describing the allocation of manufacturing processes in different factories, as an example of one of the diverse diagrams in the Airbus design model.

We developed parsers to convert the system design model from Cameo into a knowledge graph format under the SysDO schema, enabling querying with SPARQL [18]. Note that the transformation of SysML models into RDF (Abox construction) poses challenges by itself, such as model-to-ontology alignment, tool-specific serialization, and consistency checks. These challenges go beyond the scope of this paper and are not discussed here.

In Table 5, we presented the CQs in Section 2 translated into SPARQL[5]. Owing to space limitations, Table 5 only presents a subset of the CQs, specifically those with the shortest SPARQL translations. The complete queries and corresponding returns are available on GitHub[6] as well as the complete set of SysDO ontology. SPARQL enables retrieval and manipulation by structured access to the RDF file, which is the SysDO-based knowledge graph in our case. These queries assess the ontology’s capability to answer questions about activities, sub-activities, and system structures across different MORFLT layers. These queries are sent to the knowledge graphs as input (feasible on multiple software and environments, for example, GraphDB[7] as a graph database) to get output as answers to CQs. The correctness of the answers is used to evaluate the quality of the ontology. Due to proprietary restrictions on Airbus data, the full ABox can not be released to the public. However, an anonymized demo version of the parsed knowledge graph is available in the same GitHub repository.

Table 6 summarizes expected versus actual query results: In CQ2 (Hangar-level activities), the retrieval of “Hangar” shows that the SysDO-based query can identify activities with their associated locations across diverse levels (e.g., factories, hangars). In CQ5 (System resources), the identification of associated resources in final aircraft assembly, such as “Buffer 1” and “Final wing assembly”, validates the model’s ability to represent system-resource associations via *sysdo:hasLogicalSubSystemAtAllTimes*. In CQ6 (Sub-activities and durations), the extraction of sub-activities like “Trimming” and “Sheet press

[5]https://www.w3.org/TR/sparql11-query/

[6]https://github.com/PICS-LGP/SysDO-System-Design-Ontology/tree/main

[7]https://graphdb.ontotext.com/

Table 5
SPARQL Queries Corresponding to Competency Questions

CQ	Query
CQ2	SELECT DISTINCT (?sc AS ?HangarID) (?scn AS ?HangarName) WHERE { ?s a/rdfs:subClassOf iof:System. ?f a/rdfs:subClassOf sysdo:SystemFunction. ?f rdfs:label ?fn. ?s iof:hasFunction ?f. ?s sysdo:hasLogicalSubSystemAtAllTimes ?sc. ?sc rdfs:label ?scn. FILTER(str(?fn) = "Manufacturing Elementary Part") }}
CQ5	SELECT DISTINCT ?ressourceId (?r_name AS ?ressourceName) WHERE { ?f skos:prefLabel ?fn. ?f sysdo:hasFunctionPartAtAllTimes ?fc. ?systemId iof:hasFunction ?fc. ?systemId skos:prefLabel ?sysName. ?systemId sysdo:hasLogicalSubSystemAtAllTimes ?ressourceId. ?ressourceId skos:prefLabel ?r_name. FILTER(str(?fn) = "Assembly of the manufactured parts") }
CQ6	SELECT DISTINCT ?subActivityId (?name AS ?subActivityName) ?duration WHERE { ?systemId skos:prefLabel ?sn. ?systemId iof:hasFunction ?f. ?f (sysdo:hasFunctionPartAtAllTimes | sysdo:hasFunctionPartAtAllTimes/sysdo:hasFunctionPartAtAllTimes) ?fc. ?fc bfo:BFO_0000055 ?activityId. ?fc skos:prefLabel ?fcn. ?subActivityId sysdo:processPartOf ?activityId. ?subActivityId skos:prefLabel ?name. OPTIONAL { ?subActivityId sysdo:hasDuration ?duration } FILTER(str(?sn) = "Elementary metallic part manufactruing system") FILTER(str(?fcn) = "Sheet press forming") }

forming" along with their durations (e.g., "30min", "5min") demonstrates the model's capability to handle nested functional processes and corresponding attributes (e.g., temporal).

To conclude, the ontology successfully provides the expected answers to the competency questions to demonstrate that semantic traceability is improved by SysDO across system design layers. It can bridge model silos and support interoperability in SysML-based engineering tools.

6. Conclusion

We presented the System Design Ontology (SysDO), a domain-level ontology designed to bridge semantic fragmentation across operational (O), functional (F), logical (L), and technical (T) layers in complex manufacturing systems. By extending the Basic Formal Ontology (BFO) and Industrial Ontology Foundry Core (IOF-core), SysDO formalizes cross-layer relationships, ports, functions, and processes while aligning with hierarchical perspectives (global, factory, hangar, station) and addressing different perspectives (e.g., operational, technical, functional). Evaluation through competency questions (CQs) on an aeronautics manufacturing case study confirmed SysDO's ability to retrieve activities, components, and system structures across layers, demonstrating its potential to improve semantic traceability and tool-agnostic interoperability.

Despite its contributions, SysDO reveals certain limitations. Firstly, it does not cover all the layers of

Table 6
Competency Questions with Expected and Actual Query Results

CQ id	CQ content	Expected Answer	Return of query
CQ2	Which hangar has the function manufacturing elementary parts of wings?	Hangar A	HangarName "Hangar A"
CQ5	What resources are used for the final assembly system of the Wing?	Buffer 1, Final wing assembly, Stiffners Assembly, System Installation	ressourceName 1 "Buffer 1_sfHCst02_asbl04" 2 "Final wing assembly_sfHCst01_asbl05(a)_asbl05(b)_asbl05(c)" 3 "Stiffners Assembly_sfHCst05_sfHCst04_asbl01_asbl02" 4 "System Installation_sfHCst03_asbl03" 5 "Storage wings_sfHCst06_asbl07"
CQ6	What are the subactivities of sheet press forming? What is the duration of each subactivity?	Trimming (30min), Quality inspection (30min), Sheet press forming (5min), Buffer raw material (N/A)	subActivityName / duration 1 "Trimming" "30min" 2 "Quality inspection area" "30min" 3 "Buffer raw material" 4 "Sheet press forming" "5min"

the MORFLT framework. Currently, the Mission layer and Requirement layer are missing. Thus, the traceability is partial from high-level requirement analysis to technical implementations at this stage of SysDO. Secondly, some classes and axioms in SysDO need refinement, while others risk creating unnecessary redundancies. For example, the classification of processes (fort, joint, merge) remains the same for material flow and information. Additionally, the real-world deployment of SysDO relies on parsers (e.g., Cameo-to-RDF parsers) that transform models into RDF files or knowledge graphs under the SysDO schema. These parsers remain experimental and require further development and testing in different domains.

Future work will focus on refining the SysDO classes and properties. Following LOT4OCES methodology [2], we will refine process logic by establishing concise axioms to better model sequences, concurrencies, and dependencies among processes. We also plan to integrate the layers of Mission and Requirements into SysDO to formalize top-down traceability from business objectives to technical specs, aligning with MORFLT's full scope. We are also working on connecting the system design ontology to the simulation ontology, thereby improving interoperability between static design models and dynamic simulation models. We look forward to testing SysDO in various scenarios from other fields to enhance its broader applicability and generic utility.

Acknowledgements

This work is performed within the EXCELAB project, funded by the Civil Aviation Research Council (CORAC) of the French Civil Aviation Authority (DGAC). The authors express gratitude to Airbus and other aircraft manufacturer project partners (Dassault Aviation, Daher Aerospace, Airbus Atlantic and Liebherr Aerospace Toulouse) for their support and contribution.

Disclosure of Interests

Since September 2024, Mohamed Hedi KARRAY has joined the European Innovation Council and SMEs Executive Agency. The views expressed in this publication are the authors' responsibility and do not

necessarily reflect the views of the European Commission nor of the European Innovation Council and SMEs Executive Agency. The European Commission, the European Innovation Council, and SMEs Executive Agency are not liable for any consequences stemming from the reuse of this publication.

Declaration on Generative AI

During the preparation of this work, the author(s) used ChatGPT and Grammarly exclusively to: Grammar and spelling check, Paraphrase and reword. After using this tool/service, the author(s) reviewed and edited the content as needed and take(s) full responsibility for the publication's content.

References

[1] A. Kossiakoff, W. N. Sweet, S. J. Seymour, S. M. Biemer, Systems Engineering Principles and Practice, John Wiley & Sons, 2011.

[2] A. Sarkar, C. Masolo, F. A. Zaccarini, D2.9 - TLO/MLO Guidelines and Recommendations, Technical Report, ontocommons, 2023.

[3] M. Grüninger, M. S. Fox, The role of competency questions in enterprise engineering, in: Benchmarking—Theory and practice, Springer, 1995, pp. 22–31.

[4] 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:10.1002/iis2.13041.

[5] C. Ducamp, F. Bouffaron, D. Ernadote, J. Wirtz, A. Darbin, MBSE approach for complex industrial organization program, INCOSE International Symposium 32 (2022) 839–856. doi:10.1002/iis2.12967.

[6] D. A. Wagner, M. B. Bennett, R. Karban, N. Rouquette, S. Jenkins, M. Ingham, An ontology for State Analysis: Formalizing the mapping to SysML, in: 2012 IEEE Aerospace Conference, 2012, pp. 1–16. doi:10.1109/AERO.2012.6187335.

[7] H. Graves, Integrating sysml and owl, Proceedings of OWL: Experiences and Directions 2009 (2009).

[8] D. Orellana, W. Mandrick, The ontology of systems engineering: towards a computational digital engineering semantic framework, Procedia Computer Science 153 (2019) 268–276.

[9] R. Chen, C.-H. Chen, Y. Liu, X. Ye, Ontology-based requirement verification for complex systems, Advanced Engineering Informatics 46 (2020) 101148. doi:10.1016/j.aei.2020.101148.

[10] J. Lu, G. Wang, M. Törngren, Design Ontology in a Case Study for Cosimulation in a Model-Based Systems Engineering Tool-Chain, IEEE Systems Journal 14 (2020) 1297–1308. doi:10.1109/JSYST.2019.2911418.

[11] H. Wardhana, A. Ashari, A. Kartika, Transformation of SysML Requirement Diagram into OWL Ontologies, International Journal of Advanced Computer Science and Applications 11 (2020). doi:10.14569/IJACSA.2020.0110415.

[12] X. Zheng, X. Hu, R. Arista, J. Lu, J. Sorvari, J. Lentes, F. Ubis, D. Kiritsis, A semantic-driven tradespace framework to accelerate aircraft manufacturing system design, Journal of Intelligent Manufacturing 35 (2024) 175–198. doi:10.1007/s10845-022-02043-7.

[13] R. F. Calhau, T. Prince Sales, Í. Oliveira, S. Kokkula, L. Ferreira Pires, D. Cameron, G. Guizzardi, J. P. A. Almeida, A system core ontology for capability emergence modeling, in: International Conference on Enterprise Design, Operations, and Computing, Springer, 2023, pp. 3–20.

[14] R. Arp, B. Smith, A. D. Spear, Building ontologies with basic formal ontology, Mit Press, 2015.

[15] M. Drobnjakovic, B. Kulvatunyou, F. Ameri, C. Will, B. Smith, A. Jones, The Industrial Ontologies Foundry (IOF) Core Ontology, in: CEUR Workshop Proceedings, volume 3240, CEUR-WS, 2022, pp. 1–1.

[16] OMG, Ontology Definition Metamodel™ (ODM™) | Object Management Group, https://www.omg.org/odm/, 2014.

[17] S. Namaki Araghi, Z. Liu, A. Sarkar, T. Louge, M. H. Karray, Digital twin's anatomy: A cross-sector framework with healthcare validation, IEEE Access 13 (2025) 21306–21334. doi:10.1109/ACCESS.2025.3528736.

[18] J. Pérez, M. Arenas, C. Gutierrez, Semantics and complexity of sparql, ACM Transactions on Database Systems (TODS) 34 (2009) 1–45.