Core Ontology Usability: From a Formalized Knowledge Base to the Development of a System of Systems Domain Understanding

Joyce Martin *, Jakob Axelsson , Jan Carlson

Posted Date: 5 February 2026

doi: 10.20944/preprints202602.0288.v1

Keywords: core ontology; ontology formalization; domain ontology; ontology usability

 doi:10.20944/preprints202602.0288.v1

Article

Core Ontology Usability: From a Formalized Knowledge Base to the Development of a System of Systems Domain Understanding

Joyce Martin *, Jakob Axelsson and Jan Carlson

Computer Science and Software Engineering, Mälardalen University, 721 23 Västerås, Sweden
* Correspondence: joyce.martin@mdu.se

Abstract

This paper describes the step-by-step processes towards the formalization of a core ontology for missions and capabilities in a system of systems, and the development of a specific SoS domain ontology from the formalized ontology. The study adopts the SABIOx methodology to trace the different aspects of the ontology development process from requirements, setup, capture, design, and implementation phases. This study also demonstrates the core ontology's usability and reusability. Usability refers to its adequacy for specific use as a reference point for SoS knowledge exploration for development and operational purposes. Reusability refers to adequacy for several uses, such as facilitating the understanding of different domain-specific systems of systems. This is done in three steps: formalization of the core ontology, exploration of the usefulness of this formalization, and development of a domain ontology from the core ontology. These result in: the application of ontology tools to demonstrate the machine readability, inferences, and possibilities to querying the ontology knowledge base; the incorporation of systematic ontology development processes; and testing of the core knowledge with a domain prompt. An alignment of these aspects provides different points of view of how an SoS can be formulated, how the concepts collectively describe the development of an SoS emergent behavior, and how the ontology knowledge base can be explored to support decision frameworks guiding an SoS.

Keywords: core ontology; ontology formalization; domain ontology; ontology usability

1. Introduction

This study implements a formalization and usability assessment of an ontological approach to understanding System of Systems (SoS). An SoS is a system that implements its emergent behavior by leveraging the capabilities of independent constituent systems. SoS are characterized by the diversity, heterogeneity, operational and managerial independence, and the evolutionary development of their constituent systems (CS) [1]. These characteristics highlight the need for different thinking principles for working with an SoS, compared to traditional integrated systems, where systems engineering problem-solving approaches suffice. This is because an SoS brings a different perspective to interaction, interdependence, integration, and objectives of a system.

SoS are characterized by different pain-points, i.e., areas of interest that constrain SoS development and operation. Some pain-points relate to how systems engineering efforts can be applied towards SoS capabilities and requirements, complexities, interdependencies, testing, and learning. Other pain-points include SoS collaboration patterns, leadership, and thinking principles [2]. What then is a reasonable approach to create a theoretical understanding of the development and operations of an SoS? How can one create a holistic view, that governs interrelationships and processes, meanwhile allowing one to envision both the structural and dynamic aspects of an SoS? One approach is to leverage the art of systems thinking that allows one to develop a framework for analyzing the whole

 doi:10.20944/preprints202602.0288.v1

from the interaction of parts [3], knowing that *the sum of the parts is greater than the whole*. This comes with the need to facilitate the understanding of holism in a more structured manner. How can we then use systems thinking for a more generalized solution to truly formulate and explain the essence of an SoS? Addressing this question points us towards the creation of a knowledge representation for SoS, that paves the way towards creating and supporting SoS thinking principles.

In this study, we start from an ontology-driven approach to understanding SoS, i.e., a core ontology for mission and capability in SoS, as developed and evaluated by [4][5]. Ontology-driven approaches provide a shared conceptualization of a domain, and are driven by the need for consistency and conformance to existing domain knowledge and rules, and continuous evolution as knowledge evolves. A conceptualization abstracts reality, to balance truthfulness to reality i.e., *domain appropriateness*, with conceptual clarity, i.e., *comprehensibility* This balance facilitates communication and reasoning on the purpose of the ontology model [6]. Overall, an ontological approach creates a machine-readable representation that can be queried and inferred, making the ontology model more usable using ontology tools. The study looks at usability from three main aspects, syntax, semantics, and pragmatics as described by [7]. Syntax conveys the rules and structural appropriateness. Semantics conveys literal clarity. Pragmatics conveys contextual awareness. Since we are dealing with a core ontology, whose use involves the development of domain ontologies, we think of both usability, and reusability, i.e., usability as the adequacy for a specific use, and reusability as the adequacy for several uses [8]. The main contributions of this study are:

- A formal representation of a core ontology for SoS in an ontology language, and subsequent testing of constraints defined in the core ontology.
- Insights into how a formalized ontology can be queried to support the categorization and characterization of different kinds of decisions inherent in an SoS.
- A proposed mapping of the core ontology into a domain ontology, to understand the contextual usability of the ontology in a real-world SoS.

This paper is structured as follows: Section 2 gives a brief overview of the SABIOx methodology as employed in this study. The following sections describe different phases of the SABIOx methodology: Section 3 describes the requirements phase. Section 4 is on the ontology set-up phase. Section 5 is on the capture phase. Section 6 describes the design phase, and Section 7 describes the implementation phase. Section 8 discusses the findings, and Section 9 concludes the paper.

2. Methodology

As knowledge representation artifacts, ontologies go through a lifecycle of their own. Therefore, to understand the usability of the core ontology, we trace its lifecycle phases through its development process, consolidate its formalization in an ontology language, and use it to develop a domain ontology. This study employs the Extended Systematic Approach for Building Ontologies (SABiOx) [9] methodology. As a basis, SABIOx describes five phases in the lifecycle of an ontology. Each phase is described with its associated inputs, processes, and outcomes. These SABIOx phases are as described in the following sections, and seen in Figure 1. To briefly summarize the phases of this methodology:

 doi:10.20944/preprints202602.0288.v1

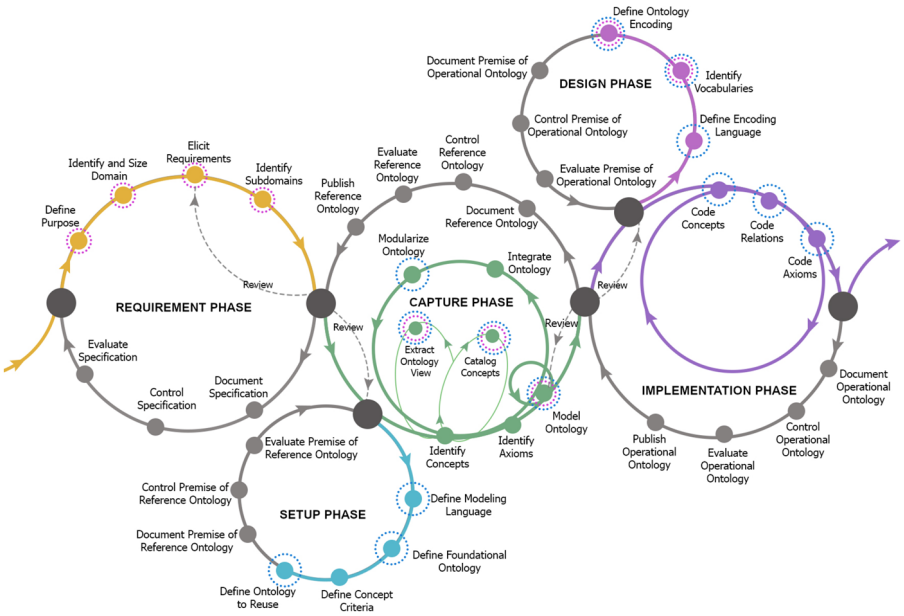


Figure 1. SABIOx methodology for ontology development, Reprinted with permission from [9]

- Requirement phase: describes *what* the ontology is about, *why*, and *what-for* it is developed.
- Setup Phase: defines questions and criteria that will guide the ontology development, and therefore, the choice of modeling-related frameworks.
- Capture Phase: identifies ontology concepts and axioms by answering the setup phase questions, and by using expert knowledge to implement desired constraints for the domain.
- Design phase: makes technology-based implementation decisions, covering the codification and presentation of the ontology in a machine-readable tool.
- Implementation phase: implements the ontology in a formal language, followed by verification and validation of its implementation.

This study gives an overview of all phases to put the ontology development in the right perspective. It must be noted that the ontology in this study is the result of several iterations of evaluation and refinement [4,5]. However, the greater focus of this study is on the design and implementation phases, whose activities include ontology formalization and extrapolation to a domain ontology. The following sections elaborate on each phase of the ontology.

3. Requirements Phase

The requirements phase seeks to identify the purpose, reasons, and the designated users of the ontology. This phase produces a specification document describing what the ontology is about, why it was developed, its purpose, and prospective users. This phase involved brainstorming, workshops, and discussions to identify what is necessary in the SoS domain. This is achieved in four main steps:

a) Purpose definition

The ontology is a core ontology for mission and capability in SoS.

- Reasons: provide an explanatory model of the SoS domain, hinged on two aspects, *mission* and *capability*, to support SoS thinking principles.
- Designated users: scientific community, and SoS practitioners interested in knowledge representation and alignment, and learning the complexity of SoS.

b) Sizing of the domain

This step focuses on setting boundaries of the ontology by defining its expected dimensions. The input of this step comes extensively from learning from application-based scenarios, from which the ontology developers can extract the core SoS domain knowledge. Dimension herein is defined as horizontal or vertical, corresponding to the uniqueness of the classes, and the depth

 doi:10.20944/preprints202602.0288.v1

of class hierarchies, respectively. These dimensions define how dispersed the classes are, and how deep the level of details is, respectively. About the core ontology, this step involved reviewing literature to understand the SoS domain, gauging dependencies among SoS concepts, and evaluating which concepts are better representations at the core level to establish the minimality of the core ontology. However, this step proved challenging as it ought to define the boundaries of a domain. In a connected world, boundaries between domains are not always clear, and therefore, we prioritized minimalism.

c) Requirements elicitation
The activities in this step involve the identification and negotiation of requirements. It highlights how the ontology team defines the functional and non-functional requirements for the ontology. Functional requirements are defined through three high-level competence questions (CQ):

- What does an SoS require?
- How does an SoS operate?
- What does an SoS realize?

Non-functional requirements are grouped in three categories: quality, design, and intended use;

- Quality requirements focus is on different *usability* aspects of the ontology.
- Design requirements are based on choosing an open-access user-friendly ontology language, and reusing existing concepts.
- Intended use requirements focus on the very purpose of the ontology i.e., scientific alignment. In this, we checked how the ontology aligns with existing foundation ontologies and ontology foundries.

d) Identification of subdomains
From the pre-defined competence questions, three keywords stand out: *requirements*, *operation*, and *realization*. The advantages of having singled out these domains include an opportunity to incorporate modularity in the ontology by explicitly outsourcing ontologies of these domains. But in our core ontology case, minimalism is key, and extensibility can be optional for a particular ontology user choices.

These steps, accompanied by documentation, control of specifications, and evaluation, result in valuable input for the setup phase.

4. Setup phase

The setup phase formulates questions and criteria guiding the ontology development. These formulations end up defining the decisions that impact the ontology modeling tasks, including: the definition of the modeling language, the conceptualization definition of what qualifies as a concept for the domain, and the choice of foundational ontology baselines and reuse of existing ontologies. The outcome of the setup phase is the reference ontology.

- The reference version of the core ontology is expressed informally in the Unified Modeling Language (UML) [10]. The choice of UML follows its simplicity and understandability in expressing relations and corresponding cardinality between concepts of the ontology.
- The reference ontology is analyzed in the light of the Basic Formal Ontology (BFO) foundational ontology. We use analogy to map between the reference ontology concept and BFO entities [**?**]. The focus of the mapping is to identify what kind of entities are contained in the reference ontology.

From the high-level CQ defined in requirements elicitation in the requirements phase, the setup phase breaks down these high-level CQ into more specific questions. This breakdown aims at highlighting what is actually part of the ontology, i.e., the SoS requirements, operations, and realization. These are summarized in the following itemization, with the actual ontology concepts italicized:

 doi:10.20944/preprints202602.0288.v1

a) SoS requirements: how the independent *systems* contribute their *capabilities*
- Which systems does the SoS contain?
- Which capabilities are provided by these systems?

b) SoS operation: formation of *constellations*, creation of *mission threads*, creation of *capability configurations* that show variations and possibilities in the SoS
- Which constellations are possible in this SoS?
- Which mission threads are addressed by which constellations?
- Which capability configuration options are possible in an SoS?

c) SoS realization: identification of *missions* possible in the specific SoS
- Which missions are part of this SoS?

5. Capture phase

The capture phase catalogues the respective ontology concepts, relations, and axioms by answering the setup phase questions, using expert knowledge to implement the respective constraints for the domain. This phase involves cataloging of concepts. For this core ontology, the capture phase involved different stages and milestones. The overall capture process was iterative, and included stakeholders' workshops which incorporated industry participation, case study analysis, and illustrations using wildfire crisis management, and road construction cases to elaborate on the essence of SoS [10].

We identified 13 main concepts for the core ontology, with some subclasses and attributes. Knowledge sources for the core ontology concepts included: engineering handbooks, standards, manuals, frameworks, and research studies. This phase further defines axioms, ontology views, and models the ontology in a technology-independent representation, applying foundational ontology, and creates an ontological commitment for describing the SoS domain. The catalogue of concepts was adapted from ISO/IEC/IEEE 21839 and 15288:2015, OMG Unified Architecture Framework Modeling Language, DoD architectural framework, Mission Engineering Guide, and IEEE 1362-1998 [10].

Preliminary efforts were made to demonstrate ontology views. Three views were identified: capability, mission and configuration views [11]. The capability view highlighted how capabilities are generated and interfaced, therefore, covering attributes such as capability definition and CS identification. The mission view highlights mission decomposition and mission thread execution, where the attributes include mission context and constraints. The configuration view highlights constellation, measures, and capability configuration operations, where the attributes included capability tradeoffs, standards, and frameworks [11]. These three views, i.e., capability, mission, and configuration views highlighted different abstraction levels corresponding to SoS requirements, planning, and implementation respectively.

The ontology was further revised. This revision aimed at further minimizing the concepts to reflect the core of SoS. This resulted in the reduction of concepts from 13 to 9 concepts. This was facilitated through an exploratory evaluation study [5]. This modified version of the ontology was benchmarked against a foundation ontology. The Basic Formal Ontology (BFO) was employed to show the kind of entities represented by the ontology concepts [12]. The core ontology is therefore a mix of different kinds of BFO entities: object, object aggregate, disposition, planned process, design specification, and plan specifications. This mapping tells the ontology user what is expected of each concept of the ontology. It also highlights how the ontology relate with entities from other ontologies and foundries.

The outcome of this capture phase is a complete reference ontology with axioms, a description in natural language, and a model dictionary. From this stage, the reference ontology concepts, relationships, and foundational identities are as summarized in Figure 2.

 doi:10.20944/preprints202602.0288.v1

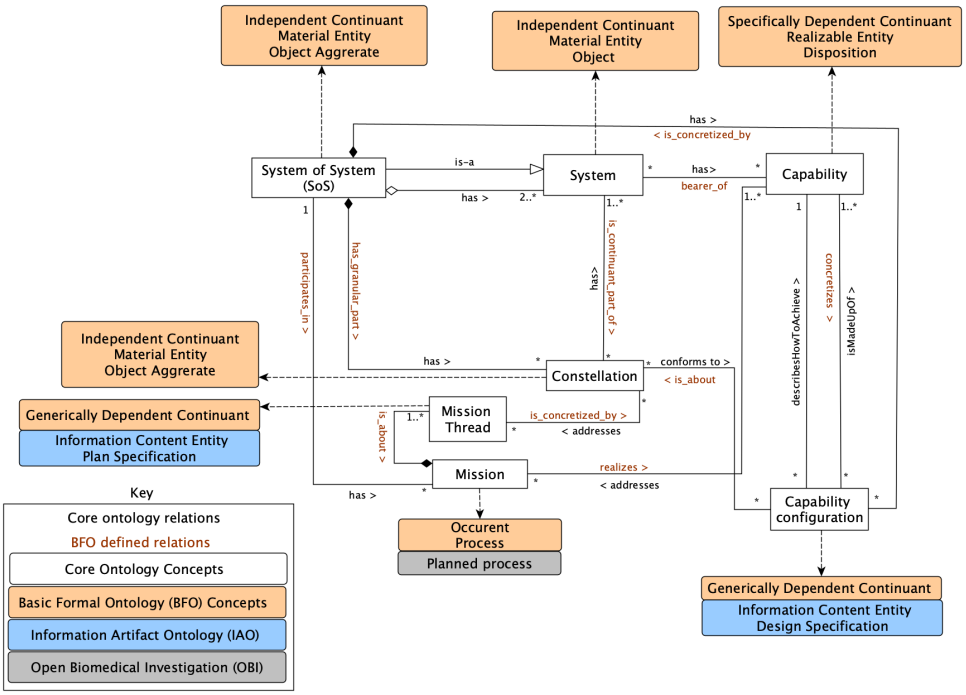


Figure 2. The core ontology with specific foundational mapping of concepts and relationships [12]

6. Design Phase

In this stage, the architecture of the formal ontology is derived from the reference ontology. This is an iterative process that sees to it that the quality standard is achieved. The input to this phase is the reference ontology as depicted in Figure 2. This phase defines technology-based implementation decisions on the operational ontology, and decides on the codification rules and guidelines. It responds to how the technological choice positively or negatively affects the requirements defined in the requirements phase. Activities involved in this phase include:

- Choice of encoding language: The requirements of an ontology language are highlighted by [13]: *well-defined syntax and semantics, efficient reasoning support, sufficient expressive power, and convenience of expression.* We opted for Web Ontology Language (OWL), which is highly popular in the ontology community. OWL reasonably fulfills the language requirements by building on the Resource Description Framework (RDF) data model and Resource Description Framework Schema (RDFS) hierarchy support, adding a mathematical logic reasoning engine. We explore and use OWL in Protégé, an ontology editing tool [14]
- Definition of ontology encoding: choice of presentation rules is based on standard encoding techniques

The design phase choices are summarized in Table 1.

 doi:10.20944/preprints202602.0288.v1

Table 1. Summary of ontology codification with respective symbols as seen in the ontology editor.

Codification Item/ Protégé symbol	Operational Ontology
Language	Web Ontology Language (OWL) with Description Logic (DL)
Core class name	Follow the pattern *Pascal Case*, with no space: *SystemOfSystem, System, Capability, CapabilityConfiguration, Constellation, Mission, MissionThread*
Domain class name	Follow the pattern *Pascal Case*: *AssessTerrain, ControlFire*
Object property name	Follow *Camel Case* pattern: e.g. *isMadeUpOf*
Data property name	Follow *Camel Case* pattern: e.g. *acccuracyLow*
Instances name	Follow *Snake Case* pattern: e.g. *move_equipment*
Class description	Classes are described using the comment property of RDFS
Specialization relationships	Defined using the *subClassOf* property of Resource Description Framework Schema (RDFS)

7. Implementation Phase

The implementation phase aims to transform the reference ontology into an operational ontology, i.e., a machine-readable version. This entails ontology encoding, verification, and validation. Ontologies are prone to different kinds of errors. These can include hierarchical errors (Taxonomy), design errors such as redundancies, or encoding errors such as incorrect syntax [15].

7.1. Encoding

The encoding process involved transforming the concepts and relationships from the UML model into an OWL model. This meant defining and differentiating bi-directional relationships between concepts, domain and ranges of these relationships, as well as their inverses. The blue-highlighted relationships are encoded with the respective domain and range i.e., as global relationships, the corresponding inverse relationships are highlighted in brown in Figure 3. Figure 4 displays the graphical visualization of the core ontology. We started with an evaluation of loops within the ontology to identify any potential errors linked to the compositional structure of the ontology. The relationships between capability and capability configuration were revised, to correctly encode the relationships between capability and capability configuration. This was necessary to differentiate the ontology's two forms of capability, i.e., *CS capability* and *SoS capability*. These were added as subclasses of the Capability concept. The OWL-implemented ontology is examined using reasoners, resulting in a consistent and coherent reasoning.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 5 February 2026 doi:10.20944/preprints202602.0288.v1

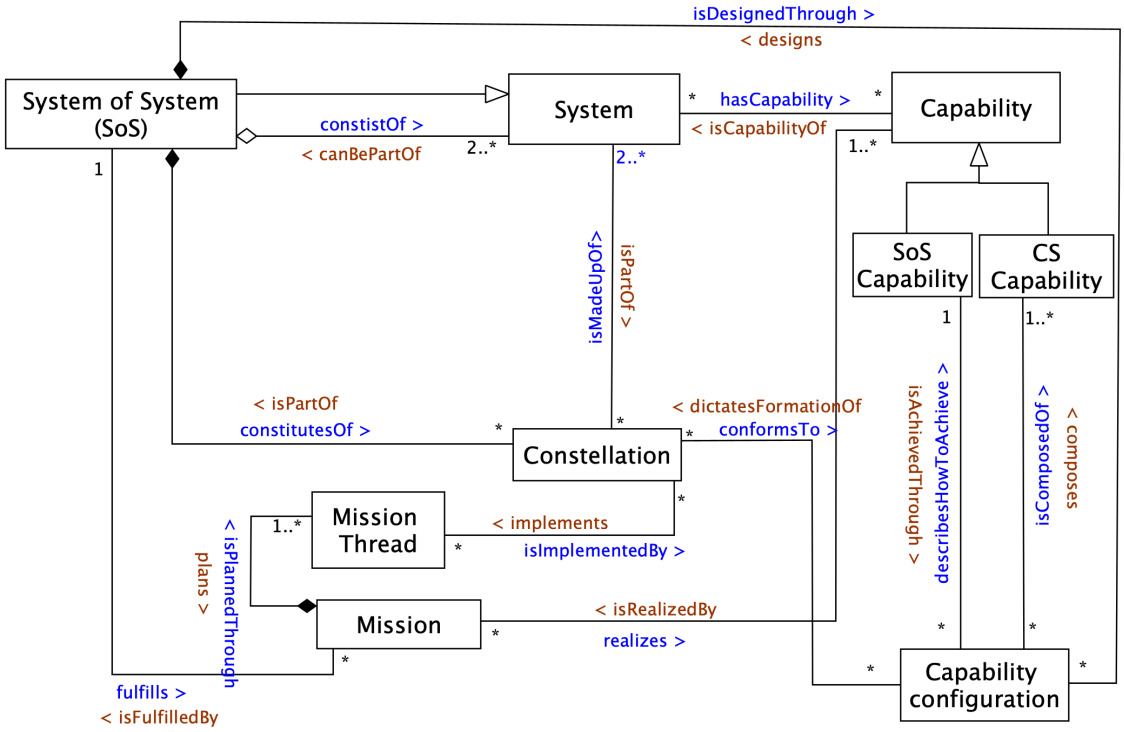


Figure 3. The encoded core ontology

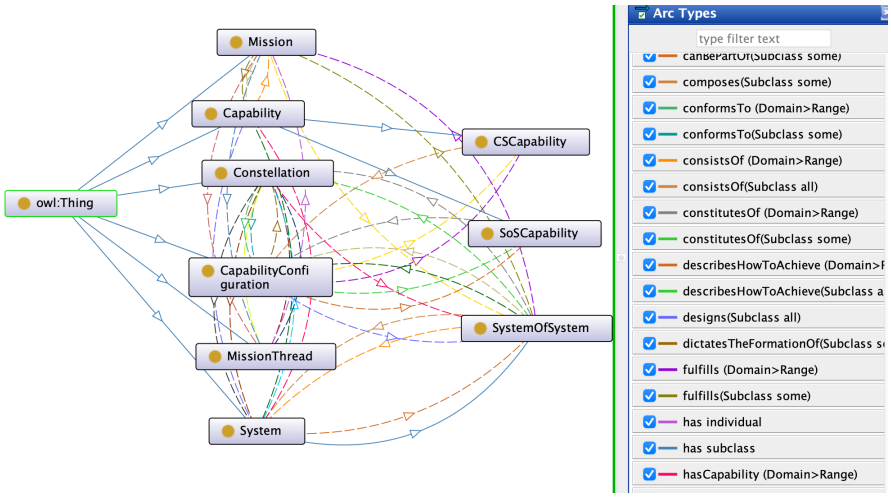


Figure 4. Graph-based view of the core ontology

Having implemented the core ontology, the next step was to verify and validate the implementation.

7.2. Verification

Ontology verification checks the internal consistency of the ontology. This includes the logical consistency, alignment, reasoning, and the correct use of ontology patterns. The SABIOx methodology suggests that: *For ontology verification, one must evaluate whether the functional and non-functional requirements raised are answered by means of queries applied on the instantiated ontology*. Since our focus is on a core ontology, not a domain ontology, we explore how we choose to view functional and non-functional requirements. We also explore and highlight various mechanical verification checks for the core ontology.

 doi:10.20944/preprints202602.0288.v1

7.2.1. Functional requirements

Functional requirements are generally defined as covering the *intended behavior of the system, what the system should perform*. In this context, our system refers to the core ontology, which is but an explanatory model of the SoS domain.

Table 2. Functional requirements as parts of ontology concepts

Competency Question	Corresponding Concepts
What are the SoS requirements?	This is extended from the concepts *system* and *capability* where a system provides a capability disposition towards a certain activity.
How does the SoS operate?	This is extended from the linkage between *capability configuration* plan, aggregation of *constellations*, and the *mission thread* step-by-step plan.
What does the SoS realize?	This is extended from how *CS capabilities* are orchestrated into *capability configurations*.

7.2.2. Non-functional requirements

Non-functional requirements generally *define how the system should perform, and the quality of respective attributes*. With the minimalist nature of the core ontology, attributes are not defined. Inadvertently, any SoS can have its own attributes, but by virtue of it being an SoS, these attributes converge towards addressing certain properties inherent in an SoS setting. In Section 7.3 below, we highlight different application-agnostic qualities from which various SoS attributes can be extracted. We relate these qualities to the foundation values of the respective concept. Therefore, for this core ontology, the non-functional requirements directly checked are linked to those mentioned in Section 3, quality, design, and intended use requirements.

The intended use is demonstrated through alignment with the foundational ontology BFO, and other ontology foundries as seen in Figure 3. This verifies its conformance to a foundation and highlights the types of entities for each concept. The design requirements are addressed through the selection of OWL-DL, an open source ontology language, and the use of Protégé, a tool for ontology editing. The quality requirements are addressed through checking the ontology's logical consistency, querying support, and the use of ontology patterns.

- Logical consistency: this focuses on the identification of contradictions. It is achieved through a consistent and coherent check using the Web Ontology Language - Description Logic (OWL-DL) reasoners. Running the reasoner on the core ontology resulted in a consistent and coherent outcome.
- Querying support: There was no inference created by the reasoner, which we attribute to the fact that the ontology is compact. OWL-DL queries resulted in the expected outcome.
- Ontology design patterns (ODPs) check: *ODPs are reusable modeling solutions that encode modeling best practices* [16]. They are agreed-upon ways of framing classes, properties, and individuals. ODP was proposed as a solution to facilitate the semantic web by offsetting the gap between the ontology development process and tools, and the application of domain expertise. ODPs are therefore prescribed best practices for ontology engineering. From the core ontology, we will consider two kinds of ODPs: logical and content ODPs [17], i.e., context-independent formal expression, and context-dependent conceptual expressions, respectively.

    a) **Transformation and partition pattern**
    The transformation pattern checks the whole-part relationships, whereas the partition pattern defines mutually exclusive entities. For the transformation, the structural check is whether the transitivity, reflexivity, and symmetry relationship observed to convey the

 doi:10.20944/preprints202602.0288.v1

required constraints. This is a logical pattern to convey the right structural make-up of the ontology.

- For the loop connecting SoS and System concepts, the core ontology defines the *constistsOf* relationship as irreflexive.
- Value partition pattern defines mutual exclusivity, i.e. whether subclasses are defined as mutually exclusive. The core ontology includes only two subclasses, *CSCapability* and *SoSCapability*, which are defined as disjoint. However, these do not necessarily form the union of the superclass; this is a conscious choice to balance the need for the core ontology to evolve to include other forms of capabilities as knowledge evolves.

b) **Role pattern**

Role pattern focuses on identifying the contextual nature of classes. It includes content OPDs that address questions: Does the ontology include roles? How are they modeled? This is to exemplify temporal behavior, which can be assumed by classes. Whether roles are classes, individuals, or properties has implications on the ontology expressiveness, simplicity, and questions that can be answered [18], [19].

- In SoS, this is particularly important as classes such as *constellations* and *MissionThread* are temporal constructions. In the context of this core ontology, the notion of roles is not explicitly stated, but it is implied. In this ontology, it can be applied in a combination by, e.g., defining capabilities as classes, individuals, or properties. It all depends on how one wants to balance their ontology. When it is defined as a class, it adds openness to the use of attributes and further characterization. When it is defined as properties, it is much simpler, but less expressive.
- Time-indexed role patterns explicitly say which classes are time-dependent. In alignment with the purpose of this ontology, entities in the operational part of the ontology present the most time-dependent nature. This means the relationships between *system*, *constellation*, and *mission thread* concepts. In its current state, there is no specific means to identify these as time-dependent.

7.3. Validation

Ontology validation focuses on the external consistency of the ontology. The SABIOx methodology suggests that: *for ontology validation, one should evaluate whether the ontology meets real-world situations by instantiating its concepts as individuals of the ontology* [9]. Studies on core ontology validation entail testing three main areas: coverage, definitions, and domain-specific testing. Throughout its development, the ontology underwent activities that iterated throughout these aspects, implementing additions, and subtractions of concepts to this working minimalist version. Definitions adhere to existing systems engineering definitions with slight modifications to fit the SoS context and the purpose of the ontology. The validation looks into two main aspects: the usability of the core ontology in a domain setting, and the use of heuristic validation metrics. Our focus in validation is on testing the external consistency of the ontology by instantiating it in a domain-specific wildfire crisis management SoS.

A typical wildfire crisis management is illustrated in Figure 5, where different possible interactions are foreseen. These include: fire detection, reporting, fire containment, and control efforts. These processes may involve different stakeholders with different managerial and operational independence, e.g., fire fighting teams, meteorology team providing inputs for the environmental factors, and civilians in settlements. It also includes the need to understand among other things, how nature and fire properties characterize the context of the scenario. Moreover, these stakeholders may be distributed, with underlying constraints, and at times even conflicting interests. These characteristics make wildfire crisis management a reasonable scenario to understand SoS complexity.

 doi:10.20944/preprints202602.0288.v1

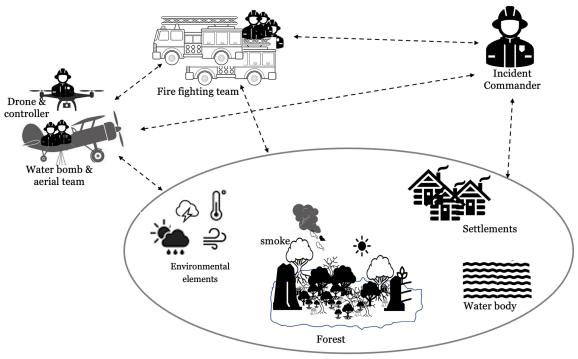


Figure 5. Typical aspects of a wildfire crisis SoS scenario [10]

7.3.1. Ontology usability

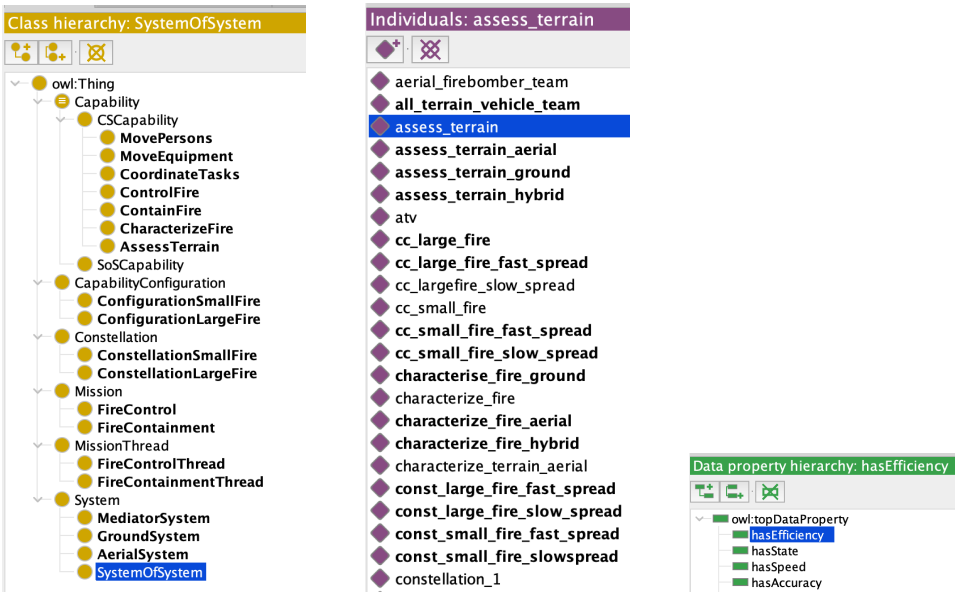
The core ontology is a minimalist representation of the SoS knowledge. To define an SoS domain ontology with clarity, embedding specific SoS expectations into the core ontology classes therefore seems necessary. We outline considerations on how one may characterize domain ontologies by thinking beyond the core ontology concepts.

With such considerations as seen in Table 3, attributes and instances can be defined to reflect the purpose or guiding factors of the SoS. A wildfire crisis management SoS domain ontology is developed by importing the core ontology, and creating sample classes, instances, and data properties. Figure 6 shows the different classes, instances, and properties for the domain ontology.

Table 3. Specific considerations corresponding to concept kinds and associated SoS characteristics

Wildfire Crisis Management SoS	**Considerations**
Fire detection constellation (object aggregate) → object states: active versus passive, incentive towards active participation in the SoS	effectiveness of the constellation
Firefighting system (object) → how a ground firefighting team versus an aerial firefighting team interface with the rest of the systems in the SoS	interfaces, boundaries, active/ passive CS in SoS
Capability (disposition) → how the capability contributions may create different tradeoffs: example moving people and equipment using a firetruck (slow) versus moving equipment with an All Terrain Vehicle (fast)	belonging, performance of the capability in a specific constellation, e.g., in accuracy
Fire detection mission (planned process) → performance measure that determines milestones towards completeness	milestones, time-factor, performance
Fire detection step-by-step procedure (plan specification) → the coordination plan for the mission context	feasibility, tradeoffs
Capability configuration, a template of possibilities (design specification) → a list of diverse combinations and outcomes towards the SoS emergent behavior	context, interoperability, reusability

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 5 February 2026 doi:10.20944/preprints202602.0288.v1



(**a**) Wildfire SoS ontology hierarchy

(**b**) Some Instances of Wildfire SoS ontology classes

(**c**) Some data properties

Figure 6. Parts of wildfire crisis management SoS ontology depicted in Protégé

In this validation, we will focus on domain-specific checks on how the ontology supports different decisions inherent in the SoS. A typical class hierarchy of the ontology is as seen in Figure 6. We test how the ontology responds to six categories of queries: why, who, what, when, how, and where. These are basic means to clarify the usability of the ontology.

- Why do we need an SoS? From the ontology, we can view SoS as originating from two main aspects: the availability of systems, and the need for missions that are SoS-worthy. The why then describes the instance of SoS Capability achievable from these systems, and the necessary classes of missions to be accomplished. Figure 7 shows two classes of *Mission* and their corresponding instances.

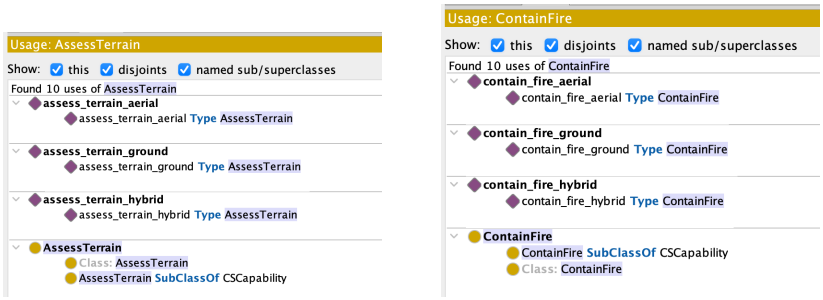
(**a**) Subclasses

(**b**) Properties

Figure 7. Sample classes and subclasses, and property assertion for a constellation for large fire.

- Who participates in this SoS? The describes the stakeholders/ ownership of the different classes of systems involved in the SoS. In this core ontology, stakeholder is not explicitly stated as a concept. Stakeholders can be implied from how one describes CS capability. In our example, CS capability is described according to their action, but they could be classified in many different ways, e.g., according to their ownerships, e.g., *control_action_municipal_council* efficiencies, whichever option demonstrates what is more important for a respective SoS, i.e., stakeholders' involvement can be embedded in their respective systems.
- What is the SoS working with? This describes the classes of capabilities needed for the SoS, and their respective instances as seen in Figure 8.

 doi:10.20944/preprints202602.0288.v1

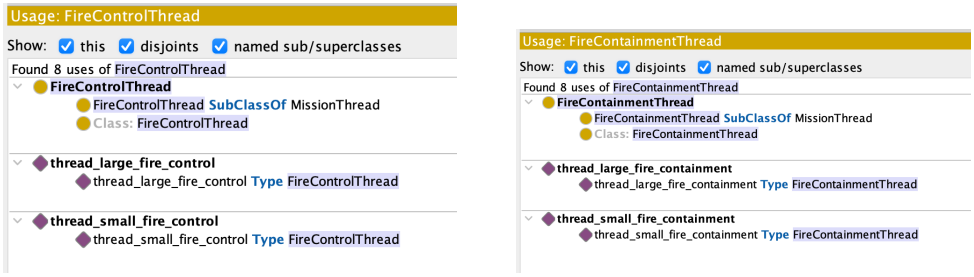


(**a**) Terrain assessment capability class

(**b**) Fire containment capability class

Figure 8. Sample of capability classes and their instances

- When? This describes the time or sequence-related nature of the SoS, showing the step-by-step mission thread. In the ontology, one is able to specify mission threads, their instances, and the corresponding constellation they are implemented by, as seen in Figure 9. However, the ontology does not yet support sequential or parallel activities.



(**a**) Fire control mission thread

(**b**) Fire containment mission thread

Figure 9. Sample mission threads and their instances

- How? This describes the operation-related nature of the SoS, linking an SoS with a capability configuration design template, and how it conforms to the constellations' actual implementation options. Figure 10 shows a sample definition of what makes up a constellation *const_large_fire_fast_spread* and its conformance to a specific capability configuration *cc_large_fire_fast_spread*.

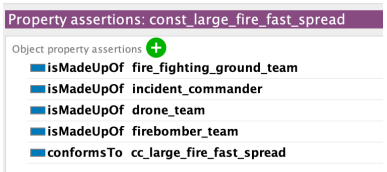


Figure 10. Typical setup of a constellation

- Where? This describes constellations formation and their adaptability. From the core ontology perspective, this is related to the context of the mission, and forms a baseline defining capability configurations.

In summary, the above highlights show the ease of use of the ontology, which is summarized as:

- Constellation is the implementor in the SoS; its adaptability must be taken in consideration in the capability configuration design plan.
- The feasibility of these configurations is observed by the context of the mission
- Constellations originate from systems; therefore, the reliability of systems as they participate in the SoS must be measured.

 doi:10.20944/preprints202602.0288.v1

- This reliability can be measured through the performance of the capabilities of these systems in the SoS.
- Missions can be traceable based on the constellations assigned and the overall relational performance of these constellations.

7.3.2. Ontology metrics

Other structural metrics can validate how acceptable and well-crafted an ontology is. This includes evaluation of the ontology hierarchy, properties, complexity, restrictions, centrality, and others, as summarized by Franco et al. [20]. Our core ontology is very small with a hierarchy depth of 3, i.e., the longest path from *thing* to *leaf*. This is on the lower end of the recommended hierarchy depth, meaning the ontology is easy to navigate through. To further check different metrics of the ontology, we employed the OntoMetrics tool. OntoMetrics is an open source web-based tool that *validates and displays statistics about a given ontology* [21]. Here we describe two metrics: relationship richness and axiom class ratio.

- *Relationship richness, 0.48: the ratio of the number of non-inheritance (NI) relationships to the total number of relationships*. It depicts the diversity of relationships in an ontology, where the more diverse the relationships, the better the conveyed information. The focus of this measure is to see that the NI relationships are fewer since they convey less value. The aim of the core ontology is to bring out deep knowledge; a low inheritance is preferred.
- *Axiom class ratio, 11.89: the average amount of axioms per class*. It speaks of the level of detail that describes the concepts of the ontology, and it also suggests the expressiveness of the ontology.

8. Discussion

This study has implemented a formalization and usability assessment of a core ontology for SoS. The formalization entailed converting the semi-formal representation of the ontology into a machine-readable version. The usability assessment explored how the core ontology can be extended to a domain ontology. The study highlights how the different phases of the SABIOx methodology were implemented throughout the ontology development lifecycle. Through these five phases several choices were made whose implications and results are discussed in this section.

8.1. OWL-DL

The design phase was preceded by the choice of codification language. OWL-DL was selected for the implementation of the formalization. This is a lower-level language that is guided through consistency and coherence checks. Although it is a good engine for facilitating machine readability, converting higher-level conceptual models to lower-level implementation is not always straightforward because of limited support for rules. OWL's underlying Open World assumption, i.e., any information that is not explicitly stated is not considered false but unknown, adds flexibility when considering that knowledge is always incomplete and can be extended further. However, it also means consistency and coherence checks are not definitive. This has both pros and cons. When we think of ontology as responsive to knowledge changes, it becomes advantageous. However, since we know that a core ontology is very specific and minimalistic to the core of knowledge, we want it to be definitive enough to direct SoS thinking principles.

8.2. Inferencing

The implementation phase involved correcting and analyzing the status quo of the ontology, and modifying it into a version that the tool can encode. This included the addition of two more classes to clearly separate the capability of independent systems' *CSCapability* from that of the SoS *SoSCapability*. The encoded implementation did not create any inferences. This led to further exploration of whether

 doi:10.20944/preprints202602.0288.v1

the ontology is over-axiomatized. We explored playing around with global settings for the domain and ranges of the object properties; however, we resorted to the realization that the ontology is compact and suitable even without inferences.

8.3. Usability

Ontology usability in a real-life SoS is tested by instantiating with classes and instances from a wildfire crisis management SoS. The competency questions were elaborated to address different issues about the SoS, and quality attributes were highlighted as means through which one can guide the choice of the types of attributes or data properties to address specific SoS concerns. The bigger question is then how useful the ontology is in supporting SoS thinking principles. The core ontology gives us a view of both tangible and intangible aspects of an SoS, and a way to plan how to go about conceptualizing an SoS, by understanding its abstract and concrete aspects, working through its descriptive plan and design specifications, concretization of abstract concepts, contextualization of roles, and realization of processes.

8.4. Methodology

The SABIOx capture phase discusses ontology integration. If our construction had reused any ontology, it would have been possible to integrate views from these ontologies in the overall construction for added clarity. However, when we look at the different conceptual domains involved in this ontology as specified in Section 3 we can see how we can build modularity in the core ontology by specializing these concepts towards better domain-specific conceptualizations. This can be the exploration of system ontologies, capability ontologies, and mission ontologies to pinpoint exactly what about these concepts stand out, and can prove useful in specific SoS domains.

8.5. SoS thinking principles

INCOSE complexity primer [22] suggests several ways to develop guiding principles to complexity thinking. These include:

- *Adapting to an ecosystem as opposed to starting from scratch:* it re-uses existing knowledge base in terms of the selection of terminologies, and seeks alignment to established knowledge bases.
- *Seek balance rather than optimization, and focus on an outcome space rather than a specific solution:* The core ontology approach to SoS has demonstrably showed us what is possible and probable. The capability configuration provides an outcome space, and the constellation aggregation varies the SoS boundaries to the chosen outcome. It seeks a balance between showing that an SoS is partly tangible and intangible, it shows the coordination and adaptation of design specification, with plan specifications. Therefore, regardless of what exists first, either the mission or the systems, one can always tailor the SoS to a satisfactory outcome according to the context and the acceptable tradeoff.
- *Adaptive thinking of interventions or influence as opposed to control or design:* It adapts one's thinking to how different constituents of an SoS can be of use, be it in a top-down or bottom-up approach; it shows that adaptive thinking can develop SoS-like uses whenever capabilities align.

9. Conclusions

This study sought to demonstrate the usability of a core ontology for mission and capability in SoS. The core ontology aims at being an explanatory model for the SoS domain hinged on the concepts, mission and capability. The study adopts the SABIOx methodology to trace the ontology from requirements to implementation. Particular emphasis is put on the formalization of the ontology knowledge base in an ontology language, and the extrapolation of the core ontology into a domain-specific SoS. From the design and implementation phases, we see how it is significant to balance

 doi:10.20944/preprints202602.0288.v1

conceptual clarity with domain appropriateness. As a core ontology, it provides a minimal way to highlight an SoS, such that it also gives room for selection of depth and breadth when expressing domain-specific ontologies by choosing a design pattern that balances the required simplicity and expressiveness. This study highlights tradeoffs and concerns when encoding ontologies, how the choice of a language, tools, and their underlying assumptions can influence logic, reasoning, and the quality attributes. Moreover, ontologies are also not very easy to understand, use, and manage, because of limited tool support and the complexity of expressing different models in machine-readable formats. Tradeoffs are inevitable, and therefore, tailoring the ontology to its purpose is a best practice; however, ensuring re-use, modularity, and conveying the right information remain very necessary. To make the ontology more usable within the ontology community, making it a part of an ontology foundry plays a significant role; therefore, future work suggests efforts to see how this core ontology may fit in the existing ontology foundries, how it can be extended by refining its concepts such as refining mission thread into steps, adding other concepts and relations, adding data properties, and including other validation opportunities by testing other domain ontologies.

Author Contributions: Conceptualization, J.M., J.A., J.C.; Methodology, discussion, analytics, J.M.; Writing—original draft, J.M.; Writing—review & editing, J.A., J.C., and J.M.; Supervision, J.A., J.C.; Project administration, J.A., J.C.; Funding acquisition, J.A., J.C. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by KKS grant no. 2020-0230.

Data Availability Statement: The complete OWL implementation is uploaded as a separate RDF file.

Conflicts of Interest: The authors declare no conflicts of interest.

Note: This article is a revised and expanded version of a paper entitled [Mapping a System of Systems Core Ontology to a Foundational Ontology], which was accepted to [MODELSWARD conference, Spain, 7-9 March 2026].

References

1. Boardman, J.; Sauser, B. System of Systems - the meaning of. In Proceedings of the 2006 IEEE/SMC International Conference on System of Systems Engineering, 2006, pp. 6 pp.–.
2. Dahmann, J. System of systems pain points. In Proceedings of the INCOSE International Symposium. Wiley Online Library, 2014.
3. Behl, D.V.; Ferreira, S. Systems thinking: An analysis of key factors and relationships. *Procedia Computer Science* **2014**, *36*, 104–109.
4. Martin, J.; Axelsson, J.; Carlson, J.; Suryadevara, J. Towards a core ontology for missions and capabilities in systems of systems. In Proceedings of the 2023 18th annual system of systems engineering conference (SoSe). IEEE, 2023, pp. 1–7.
5. Martin, J.; Axelsson, J.; Carlson, J.; Suryadevara, J. Evaluation of Systems Engineering Ontologies: Experiences from Developing a Capability and Mission Ontology for Systems of Systems. In Proceedings of the 2024 IEEE International Symposium on Systems Engineering (ISSE). IEEE, 2024, pp. 1–8.
6. Guizzardi, G.; Ferreira Pires, L.; Van Sinderen, M. An ontology-based approach for evaluating the domain appropriateness and comprehensibility appropriateness of modeling languages. In Proceedings of the International Conference on Model Driven Engineering Languages and Systems. Springer, 2005, pp. 691–705.
7. Ma, X.; Fu, L.; West, P.; Fox, P. Ontology usability scale: context-aware metrics for the effectiveness, efficiency and satisfaction of ontology uses. *Data Science Journal* **2018**, *17*, 10–10.
8. Russ, T.; Valente, A.; MacGregor, R.; Swartout, W. Practical experiences in trading off ontology usability and reusability. In Proceedings of the Proceedings of the Knowledge Acquisition Workshop KAW99, 1999.
9. Aguiar, C.Z.; Souza, V.E.S. SABiOx: the extended systematic approach for building ontologies, 2024.
10. Martin, J.; Axelsson, J.; Carlson, J.; Suryadevara, J. Towards a core ontology for missions and capabilities in systems of systems. In Proceedings of the 2023 18th Annual System of Systems Engineering Conference (SoSe). IEEE, 2023.

 doi:10.20944/preprints202602.0288.v1

11. Martin, J.; Cederbladh, J. Toward Formalizing a Systems of Systems Core Ontology for Capability Configuration: A SysML Approach. In Proceedings of the Conference on Systems Engineering Research. Springer, 2024, pp. 27–34.
12. Martin, J.; Axelsson, J.; Carlson, J. Mapping a System of Systems Core Ontology to a Foundational Ontology. In Proceedings of the Accepted MODELSWARD 2026, 2026, pp. 1–8.
13. Antoniou, G.; Harmelen, F.v. Web ontology language: Owl. In *Handbook on ontologies*; Springer, 2009; pp. 91–110.
14. Horridge, M.; Knublauch, H.; Rector, A.; Stevens, R.; Wroe, C. A practical guide to building OWL ontologies using the Protégé-OWL plugin and CO-ODE tools edition 1.0. *University of Manchester* **2004**.
15. Fahad, M.; Qadir, M.A.; Noshairwan, M.W. Ontological Errors-Inconsistency, Incompleteness and Redundancy. In Proceedings of the ICEIS (3-2), 2008, pp. 253–285.
16. Presutti, V.; Blomqvist, E.; Daga, E.; Gangemi, A. Pattern-based ontology design. In *Ontology Engineering in a Networked World*; Springer, 2011; pp. 35–64.
17. Gangemi, A.; Presutti, V. Ontology design patterns. In *Handbook on ontologies*; Springer, 2009; pp. 221–243.
18. Blomqvist, E. Ontology patterns: Typology and experiences from design pattern development. In Proceedings of the Proceedings of the Swedish AI Society Workshop. Uppsala, 2010, pp. 55–64.
19. Blomqvist, E.; Gangemi, A.; Presutti, V. Experiments on pattern-based ontology design. In Proceedings of the Proceedings of the fifth international conference on Knowledge capture, 2009, pp. 41–48.
20. Franco, M.; Vivo, J.M.; Quesada-Martínez, M.; Duque-Ramos, A.; Fernández-Breis, J.T. Evaluation of ontology structural metrics based on public repository data. *Briefings in bioinformatics* **2020**, *21*, 473–485.
21. Poppe, M.; Lichtwark, M. OntoMetrics. https://ontometrics.informatik.uni-rostock.de/, 2016. Accessed: January 8, 2026.
22. McEver, J.; Sheard, S.; Cook, S.; Honour, E.; Hybertson, D.; Krupa, J.; McKinney, D.; Ondrus, P.; Ryan, A.; Scheurer, R.; et al. A complexity primer for systems engineers. *International Council on Systems Engineering* **2015**.