Bridging the Semantic Web and Model-Based Systems Engineering with the Ontological Modeling Language

Maged Elaasar[1], Bentley Oakes[2], Eduard Kamburjan[3,4], Mohammad Hamdaqa[2], and Abdelwahab Hamou-Lhadj[1,5]

[1] NASA Jet Propulsion Lab, California Institute of Technology, Pasadena, USA
maged.e.elaasar@jpl.nasa.gov
[2] Polytechnique Montréal, Montréal, Canada
bentley.oakes@polymtl.ca, mhamdaqa@polymtl.ca
[3] IT University of Copenhagen, Copenhagen, Denmark
eduard.kamburjan@itu.dk
[4] University of Oslo, Oslo, Norway
[5] Concordia University, Montréal, Canada
wahab.hamou-lhadj@concordia.ca

Abstract. The Ontological Modeling Language (OML) is an ontology description language that stems from applying the principles of ontological modeling in the systems engineering (SE) field. SE has specific requirements on tooling and methodology that have hampered the application of Semantic Web technologies in this field. In particular, we identify as challenges the lack of rigorous semantics for SE models, and the lack of a well-established methodology for effectively using ontologies in SE. We describe OML principles through an SE example and describe an OML methodology and tooling. We also describe the status of OML adoption and usage, including multiple projects in academia, industry, and government. We aim to describe OML in practice, its principles, and its tooling for the semantic web community, such that ontology best practices can be transferred more easily to system engineering.

Keywords: Systems engineering · Ontology construction · Semantic reasoning · Domain-Specific Modeling Languages

1 Introduction

Model-Based Systems Engineering (MBSE) is an approach to the formulation, development, and maintenance of systems in interdisciplinary projects [44]. It sets itself apart from prior approaches to systems engineering (SE) by employing *models* and formal languages as central artifacts. Instead of documents, models are the first-class entities for representing the system, related information, and processes. These models thus formalize and/or support requirements, design, analysis, verification, and validation steps during the development of complex systems and system of systems [31]. MBSE is a cornerstone of digitalization in

engineering, and there exists a rich and growing ecosystem of languages (e.g., SysML [3]) and related tools. Studies have shown that the usage of MBSE reduces the time and cost of SE, and improves communication between stakeholders [17], despite persisting challenges in MBSE's adoption [8].

MBSE Ontologies and Challenges. This article focuses on the use of ontologies as one kind of model used in MBSE, where they are used to describe the system, its design, or to connect the system model with semantically enriched data or provenance [54]. The complex systems in MBSE require dynamic and rich connections which match well to semantic technologies. In particular, these complex systems and their development processes need to relate and integrate information across different domains. This information must then be reasoned about to ensure consistency and to infer new knowledge. Ontological techniques are also well-suited for reporting on the MBSE process and artifacts, with enhanced traceability over standard modeling approaches. Integration in MBSE using ontologies is performed using different methodologies, yet structured investigation into their advantages and disadvantages is lacking [40].

Indeed, from our experience, the *languages and frameworks currently used for ontology engineering are not suited to be integrated in MBSE workflows.* We identify the following two broad challenges, further described in section 2.

1. Ontological languages such as OWL are *not the proper level of abstraction for describing SE.* OWL is too complex and flexible for systems engineers, who have no background in knowledge engineering and wish to rely on well-known patterns (e.g., composition, aggregation) to reason about their system in a closed-world fashion with a clear interface to open-world modeling.
2. Another challenge has been the *lack of robust ontological modeling tools* suited for domains where the subject matter experts are used to textual and modular languages. Hildebrandt *et al.* report that they "have identified that there is no suitable tool support to assist domain experts in modeling ontolog[ies]. For example, tools such as Protégé [34] require significant experience in knowledge representation and semantics" [20].

Ontological Modeling Language. To address these challenges, the Ontological Modeling Language (OML) is a NASA Jet Propulsion Lab initiative to develop a textual language on top of OWL and the Semantic Web Rule Language (SWRL) to provide the necessary intermediate structures for MBSE uptake of ontological modeling. The approach of OML allows ontologies and knowledge graphs to be built constructively starting from the domain concepts using domain *vocabularies.* These vocabularies are then the basis for the integration of multiple domain-specific editors and tools.

OML offers the benefits of ontological modeling with its emphasis on interoperability and logical semantics, including composition of languages, multi-classification of instances, inference rules, and consistency checking. However, the reference implementation of OML is based on the proven and familiar modeling language development technologies of (1) the Eclipse Modeling Framework

(EMF) and (2) the VSCode stack. OML is supported by an open-source toolkit openCAESAR [9] including an Eclipse-based IDE, a Gradle-based ontology engineering workflow automation, and tools to integrate with OWL and SWRL. Common-place software engineering tools and practices such as Git and CI/CD are also employed in openCAESAR to ensure that languages and models can be versioned, federated, distributed, and verified as required.

Paper Contribution. The main contributions of this work are (1) *a description of OML*, including its underlying design principles, methodology, and tooling which brings semantic web benefits while still matching MBSE practice requirements, and (2) a report on the OML uptake in research and industry.

2 Background, Challenges, and Related Work

Modeling and Ontologies in MBSE. At its core, systems engineering is about handling uncertainty, risk, and complexity during the interdisciplinary design of complex systems [49]. This creates a tension in the very nature of its system models: On one side, there is a concrete design or system, which is suitable for *closed-world reasoning* as missing components are indeed absent. On the other side, the model is used for communication between different disciplines and is placed in the context of numerous documents and domains. Thus it must be amendable for *open-world reasoning*.

In this tension, MBSE tooling tends to support closed-world reasoning, with SysMLv2 [3] becoming established as the standard modeling language and tool ecosystem for MBSE. SysML is a textual and visual language for representing system structure and behavior in different diagram types. Semantic technologies, e.g., the RDF/OWL technology stack, tend to the open-world reasoning side of the tension, with closed-world and open-world semantics being handled by different technologies (such as SHACL) or configuration (such as in SPARQL).

In contrast to semantic technologies, the formal semantics of SysMLv2 is expressed internally in terms of its meta-model, not externally through the entities and concepts it models. Thus, many approaches have been tried to bridge the gap between open-world knowledge engineering and closed-world system modeling [41]. Techniques to maintain consistency and perform reasoning on SysML using semantic technologies, such as description logic reasoners [36], rely on giving SysMLv2 additional semantics by generating a knowledge graph explicitly.

To bring the benefits of the semantic web to SE (integration, traceability, reasoning, etc.) without employing SysML, there has been work on applying ontological techniques to SE [54]. However, we find that the two challenges previously mentioned are still present and continue to hinder ontological MBSE.

Challenge 1: Abstraction Mismatch. One fundamental issue with the application of ontological techniques is the *mismatch between the principles of SE and those of ontology languages* in the semantic web, such as the level of abstraction.

In particular, there are no intermediate layers in OWL: The meta-model of OWL2 has only two language layers: *ontologies*, and *axioms*. While axioms can

be syntactically grouped, e.g., in the functional syntax, this is barely a convention and is not exploited in textual editors. One consequence is that modules in ontologies are defined in terms of inferences and have no agreed upon semantics [29], while modules used in SE are defined in terms of encapsulation and interfaces. Note that the frame-based syntax of OWL1 was dropped specifically to enable new languages to build on top of the more fine-grained OWL2 [13].

This lack of intermediate structure also hampers the development of tool supportsuch as IDEs, as the meta-model of OWL is simply not rich enough to enable reuse and hierarchical structuring in the same way languages for system and software modeling do. This becomes especially apparent when OWL is used together with other languages, such as SWRL, or when open-world and closed-world reasoning are used in different applications on the same ontology.

To exemplify this challenge, we point to Hennig *et al.*, who provide a detailed discussion on the application of an ontology for satellite design [19]. The ontology was manually created in the OWL2 ontology format from a UML system model. While this approach allowed for consistency and verification checks, they identified concrete issues with employing OWL2: a) closed-world checks could not be applied, b) data could be modeled in the A-Box which was out-of-scope of the T-Box, c) there were no built-in part-of/containment/aggregation relationships, and d) there was difficulty reasoning about numeric values.

Challenge 2: Inadequate Tooling. The biggest hindrance to adopt open-source semantic web tooling is that *most tools are for OWL, and other languages on a low abstraction level, in isolation.* Integration of constraints, queries, mappings, *etc.* with an ontology requires manual maintenance of artifact dependencies in different languages [26] and handling of subtle differences in tool semantics [21].

More precisely, as these tools operate at the level of *logical axioms*, users must be familiar with the subtle differences that positive and negative axioms have on the *open-world semantics* of ontological languages. In particular, the open-world semantics of OWL reasoning does not map exactly with the *closed-world semantics* of a meta-modeling-based approach [18,41].

Other tools have attempted to bridge this meta-modeling/ontology gap. However, some appear to have been dormant for at least a decade, including the TwoUse Toolkit [50,51] and the NeOn Toolkit [16]. One more recent approach is OntoUML [15] which focuses on *conceptual modeling* through the use of Unified Modeling Language aligned with the top-level *Unified Foundational Ontology* (UFO). While the available tooling does address some SE requirements, it lacks the ability to build system ontologies incrementally starting from domain concepts and with custom viewpoints, as it requires an explicit alignment to UFO.

When it comes to integrated approaches, these tend to be specific to applications or domains, for example the Ontology Development Kit (ODK), ROBOT and other tools in the OBO Foundry ecosystem. No such ecosystem exists yet for connections to meta-modeling. The last group of tools focuses on visualization, such as RDF graph visualization [1] or OWL visualization [30]. Visualization in MBSE is common, but relies on the physical structure of the underlying system to be natural and is known to be a hindrance in certain circumstances [25].

Addressing Challenges with OML. The Ontological Modeling Language (OML) is part of a solution to these challenges for employing ontological techniques in MBSE. At its heart, OML has three principles: *modularity, abstraction and extensibility*. To this end, it introduces a module-system with tight control over closed-world and open-world semantics within the same language. The notion of module is based on interfacing modules from software development [53]. The modules have a standard import mechanisms for extensibility. Furthermore, the separation of modules into different classes enables to use imports without requiring knowledge about eventual disjointness axioms, as these are only possible in *bundle modules*. We will illustrate this notion of extensibility shortly.

Inside the modules, OML uses a frame-based syntax to declare (and extend) OWL entities, such as properties, classes, data types etc. Each module can be mapped to a set of OWL axioms. Different modules allow definitions of different axiom kinds, e.g., *vocabularies* cannot declare individuals, only *descriptions* can.

OML is based on the EMF technology stack, an established technology for language development. As a modeling language, OML is thus designed to improve the ergonomics of ontology authoring within a model-centric paradigm [47]. OML is thus conceptually between a *knowledge representation* language (to *describe* systems) and a *domain-specific modeling language* (DSML) framework (to *proscribe* systems). That is, OML vocabularies can be utilized as DSMLs for systems, such that a system model encodes concepts from the problem domain [28]. This tailoring allows domain experts to reason about their model more effectively without having to become experts in the solution space [27,46].

3 OML Concepts and Language by Example

We introduce a selection of OML using the Kepler16b (`https://www.opencaesar.io/kepler16b-example/`) example. It models a hypothetical space mission with the vocabulary needed to describe space missions, and then uses it to describe the concrete mission itself. A detailed OML language description, including formal grammar and mapping to OWL/SWRL, an EMF meta-model and tutorials, is found at `https://www.opencaesar.io/oml/`.

An OML ontology consists of a set of modules, which are either a *vocabulary*, or a *description*. These work analogously to modules in programming languages: They enable import and export between declared properties and entities. Vocabularies and descriptions can also be declared as *bundles*, where everything declared in this bundle then assumes closed-world reasoning as discussed in OWL terms in Section 3.3. Note that an OML ontology's semantics is a set of OWL(-DL) axioms (classes, object and datatype properties, named and anonymous individuals) and a (DL-safe) set of SWRL rules, such that OML is equivalent to an OWL-DL subset and can be reasoned about (e.g. Pellet). OML does not support features such as negative assertions and property expressions, which based on prior experience were not used for MBSE modeling.

Vocabularies An OML vocabulary model declares abstractions of OWL properties, OWL classes and SWRL rules.

Descriptions An OML description model declares abstractions of OWL named and anonymous individuals and their relations and data.

A vocabulary can *extend* another vocabulary, or *use* a description. A description can *extend* another description, or *use* a vocabulary. Bundles can additionally *include* other bundles. In any case, all declarations of the imported module become available in the importing module. There is no export or interface mechanism: All declarations are exported, there is no information hiding.

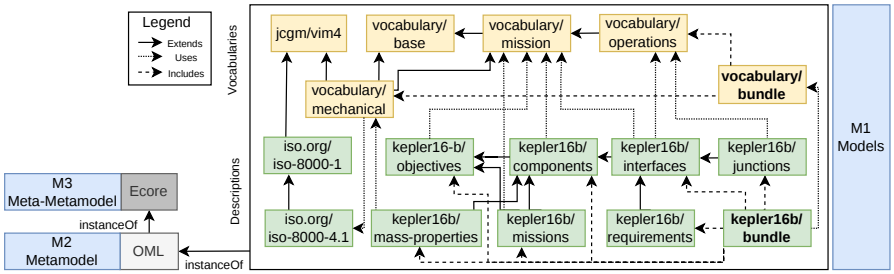

Fig. 1. Kepler16b modeling levels in OML

Figure 1 relates the *vocabularies* and *descriptions* in the Kepler16b example. Vocabularies mostly *extend* each other, while descriptions mainly *use* relevant vocabularies. Boxes labelled as *bundles* refers to a kind of OML ontology that only *includes* (i.e., imports) other ontologies. Figure 1 also shows how OML is implemented with the Eclipse Modeling Framework (EMF) in terms of *metamodels* in model-driven engineering terminology [52]. OML is defined with an (M2) metamodel that is an instance of EMF's Ecore (M3) meta-metamodel. This means that OML modules are models, i.e., M1 instances of that M2 metamodel.

3.1 OML Vocabularies

An OML vocabulary introduces classes, object properties, data types, and data properties, and their relations. It allows declaring different types and properties. Each declaration corresponds to multiple OWL axioms. We give the most important types here, and omit the introduction of scalars (which define OWL data types) and unreified properties. Figure 2 shows a part of the `mission` vocabulary of the Kepler16b example, slightly modified to illustrate more features of OML.

Concepts and Aspects. A *concept* is the OML abstraction over OWL classes. The only mandatory part of its declaration is a fresh IRI. It has four optional parts: (1) The class can be defined as an *enumeration* of imported individuals, (2) the class can refer to properties and relations as *keys*, and it can be defined using an axiom sublanguage akin to OWL Manchester syntax [22] as a (3) *specialization* or (4) *equivalence* of another entity.

OML

```
vocabulary <.../mission#> as mission {
extends <.../base#> as base
concept Objective < base:IdentifiedThing, base:AggregatedThing [
    restricts all base:aggregates to Objective ]
concept Requirement < base:IdentifiedThing
aspect SpecifiedThing
relation entity Specifies [ from Requirement  to SpecifiedThing
    forward specifies  reverse isSpecifiedBy functional ]
rule Junction-infers-Connection [
    presents(c1, i1) & joins(i1, i2) & isPresentedBy(i2, c2)
    -> connectsTo(c1, c2)]}
```

OWL

```
Class(Objective)
SubClassOf(Objective base:IdentifiedThing)
SubClassOf(Objective base:AggregatedThing)
SubClassOf(Objective ObjectAllValuesFrom(base:aggregates Objective))
```

Fig. 2. Illustration of OML vocabularies using the mission module.

An *aspect* is similar to a concept, but the idea behind aspects is to introduce reuse through a notion of mixin [4,33]: Aspects can be used as the domain of a property, and consequently a concept specializing an aspect automatically inherits the capability to be in the domain of the property in question. In the OWL mapping of OML vocabularies, aspects result in the same axioms as concepts. However, in the closed-world semantics, i.e., vocabulary bundles, they are treated different and excluded from the generation of additional disjointness axioms.

Figure 2 (top) contains the declarations of several concepts and aspects. The declaration of `Objective` is equivalent to the OWL axioms in Figure 2 (bottom).

Relation Entities. A *relation entity* is the OML abstraction over *reified* OWL object properties. A relational entity has a fresh IRI as its only mandatory element, and additionally allows to specify the domain and range by referring to entities (i.e., aspects, concepts, and expressions of the aforementioned axiom language). Additionally, a relation entity can define IRI for both the forward and backward direction. Each results in a distinct object property, which are declared to be each other inverses. Finally, a relation entity has the usual tags for its forward direction (functional, symmetric, etc.). Analogously to concepts and aspects, a relation entity can be declared to be a specialization or equivalent of another relation entity, using an axiom sublanguage. Figure 2 contains the declaration of `Specifies`. OML introduces a OWL class for reification and a corresponding SWRL rule to realize it. Figure 3 is a subset of the axioms generated.

Scalar Properties and Rules. A *scalar property* is the OML abstraction over OWL data properties. It is mostly analogous to relation entities, but without the

```
OWL
1 ObjectProperty(specifies) ObjectProperty(isSpecifiedBy)
2 ObjectPropertyDomain(specifies Requirement)
3 ObjectPropertyRange(specifies SpecifiedThing)
4 InverseObjectProperties(specifies isSpecifiedBy)
5 Class(Specifies) //for reification, rules omitted
6 ObjectProperty(specifiesFrom) ObjectProperty(specifiesTo)
```

Fig. 3. Subset of the axioms generated from the declaration of `Specifies` in Figure 2.

```
OML
1 scalar property hasId [ domain IdThing  range xsd:string  functional ]
```

Fig. 4. Example definition of a scalar property in OML.

ability to declare inverses and with less possible tags. A function scalar property from the `base` vocabulary of the Kepler16b example is declaredin Figure 4.

A *rule* has two conjuncts, which form the antecedent and consequent of a rule, with the expected SWRL semantics. The conjuncts are formed of predicates, which can refer to SWRL-builtins, builtins declared in OML or be predicates over concepts, relations and properties. Figure 2 contains one rule declaration.

3.2 OML Descriptions

An OML description describes individuals using the concepts, entity relations and properties declared in OML vocabularies. We introduce different kinds of declarations below. Figure 5 gives a description from the Kepler16b example.

A concept instance is defined by an IRI and at least one concept name. Optionally it can contain assertions about its object properties and properties. Figure 5 contains several concept instance declarations. The declaration of the two instances in `objectives` is equivalent to the OWL axioms in Figure 6.

```
OML
1  description <.../requirements#> as requirements {
2      uses <.../mission#> as mission
3      extends <.../interfaces#> as interfaces
4      relation instance data-sys-out-req : mission:Specifies [
5          from orbiter-ground-data-system-command-to-spacecraft
6          to interfaces:orbiter-data.presents.commandOut] }
7  description <.../objectives#> as objectives {
8      instance characterize-atmosphere : mission:Objective [
9          base:hasId "0.01"
10         base:aggregates characterize-liquid-ocean]
11     instance characterize-liquid-ocean : mission:Objective }
```

Fig. 5. Illustration of OML description using the requirements and objectives modules.

```
OWL
1 //for the requirements description
2 ClassAssertion(mission:Specifies data-sys-out-req)
3 ObjectPropertyAssertion(specifies
4                 orbiter-ground-data-system-command-to-spacecraft
5                 interfaces:orbiter-data.presents.commandOut)
```

```
OWL
1 //for the objectives description
2 NamedIndividual(characterize-atmosphere)
3 ClassAssertion(mission:Objective characterize-atmosphere)
4 DataPropertyAssertion(base:hasId characterize-atmosphere "0.01")
5 ObjectPropertyAssertion(base:aggregates ...)
6 NamedIndividual(characterize-liquid-ocean)
7 ClassAssertion(mission:Objective characterize-liquid-ocean)
```

Fig. 6. OWL equivalent of Figure 5. The reification of the relation instance is derived using the SWRL rules generated from the declaration of Junction as a Relation Entity.

A relational instance connects two concept instances using a *relational entity*, which corresponds to a *reified* property – a relational instance declaration instantiates this reification. Figure 5 contains one relational instance declaration for `data-sys-out-req`, which is is equivalent to the OWL axioms in Figure 6.

Both concept and relational instances can be anonymous, namely if chains are used in the assertion defining some named concept or relational instance.

3.3 OML Bundles – Open-World versus Closed-World

In the open-world assumption of OWL, if a fact is not stated then its truth value is unknown. To control this assumption, axioms can be added to explicitly rule out certain facts. In OML, this is done by importing vocabularies and descriptions into *bundles*. A *vocabulary bundle* does not add new declarations, instead it adds OWL disjointness axioms between all classes that do not have a common subclass. It does so by iterating over all classes declared in the vocabularies, as well as calculated class expressions that occur. A *description bundle* introduces enumeration axioms that state that classes contain only those individuals explicitly declared and asserted to be their member. Thus, OML uses the open-world assumption for extensibility in vocabularies and descriptions, and a closed-world assumption for validation in bundles. This is an improvement over prior SE approaches with OWL, which added closed-world axioms prematurely and limited reuse [19] Aspects differ from concepts, by being *excluded* from the generation of the disjointness axioms for the closed-world semantics, see below.

3.4 OML References and Annotations

There are some further model elements worth discussing: References and annotations. *References* allow the user to add statements to already declared model

OML

```
vocabulary ... as Signing {          //creates new vocabulary
    extends <...> as obj            //imports a vocabulary
    annotation property signedBy    //mandatory declaration
    @signedBy "BOakes"             //annotation statement
    ref concept obj:Objective }      //reference
```

OWL

```
AnnotationProperty( :SignedBy )
AnnotationAssertion( :SignedBy :Objective "BOakes" )
```

Fig. 7. Illustration of annotations and references in OML. Concept `Objective` is declared outside of `Signing` and only referenced here to add an annotation statement.

elements. *Annotations* corresponds directly to OWL annotation properties. They can be declared as part of a vocabulary, and then added in descriptions and vocabularies. Figure 7 shows a vocabulary that defined an OML annotation property `signedBy`, and uses a reference to the concept `Objective` to annotate it. The OWL axioms are again given in Figure 7.

4 OML Methodology and Tooling

OML and its toolkit openCAESAR [9] are not specific to a given development methodology, but here the methodology of JPL that guided and motivated OML in systems engineering is presented. We also briefly touch on the tooling aspects.

4.1 Development Methodology and Roles

The development of OML models involves several developer and stakeholder roles, due to the interdisciplinary nature of systems engineering. Each of the roles has specific requirements on the language and tooling.

On a high-level, development itself is split into two parts: First, a DSML [27] in the form of OML vocabularies is developed. This DSML encapsulates very specific, mission or application area-specific knowledge and must incorporate ontology engineering best practices, as it is similar to a application-level ontology. Second, a *system model* in the form of OML descriptions is developed. System models are specific to a certain stage in the system design. Here, OML is used closer to standard SE best practices.

At the start of the system modeling project, a *DSML developer* uses an OML authoring tool to define the domain vocabularies. This may involve the creation of new vocabularies (such as the Kepler16b ones), reuse of another vocabulary, or the re-implementation of an existing ontology in OML. The created vocabularies are checked for consistency and then versioned and stored in repositories.

Once these initial vocabularies have been created, they form the integration backbone throughout the project. Note that the DSML developer may also wish

(but not required) to align these vocabularies with upper ontologies such as BFO [2] and UFO, for semantic interoperability. The DSML developer may also create new authoring tools and/or adapters to connect to existing tools. Further analyses and reports may fall under the responsibilities of the DSML developer. They will then iterate on the vocabularies together with the system modelers and other stakeholders to fulfill the necessary and desired system requirements.

The *system modeler* is a classical systems engineering role tasked with developing the system itself. That is, they are primarily focused on developing the *descriptions* of the system. This may be using the same authoring tool as vocabulary development, or other modeling tools through the use of adapters.

Along with the DSML developer, the system modeler may be involved in defining further analyses or reports for the system. They will also likely have feedback for the DSML developer to improve the vocabularies used. Cross-role collaboration must be supported, such as report creation, support for versioned dependencies in repositories, and use of standard collaboration mechanisms for models such as pull requests and Git branches for the development process.

Stakeholders. Systems engineering is a process-heavy discipline [49,5,6,39]. Each system model, DSML, *and design process* must also go through reporting, review and audit processes by additional *stakeholders*. The exact nature of additional stakeholders, such as management, compliance personnel, and outside collaborators, depends on the specifics of a given project. These stakeholders likely do not have the technical knowledge required to understand the model files themselves. Thus, MBSE approaches can offer web-based reporting for stakeholders to quickly understand the system development status and the impact of changes.

For example, openCAESAR development is ongoing to be able to not only view a report for a particular Git branch of the project, but also to view the *differences* between reports on different branches. This will allow stakeholders to better understand the impact of a proposed change at the appropriate level of abstraction. The very same tools can be used for technical audits as well.

4.2 Tooling

There are four different groups of tools that must be connected: 1) authoring of OML models, 2) adapters to other formalisms, e.g., from SysML and to OWL, 3) analyses, and 4) report generators. In the OML ecosystem, openCAESAR serves as the primary tooling framework [9]. openCAESAR has been developed mainly by NASA JPL, but is a open source project with contributions from Ford, Airbus, DLR, Okean Solutions, CEA-List and the Technical University Munich.

Authoring. openCAESAR provides Rosetta, an Xtext [10]- and Eclipse-based IDE for authoring OML ontologies. Figure 8 demonstrates Rosetta OML editor, showing the editing of the *missions* vocabulary using its textual syntax, and its graphical visualization using a Sirius-based [7] diagram viewpoint. For users of VSCode IDE(s), the Luxor[38] extensions also enables to author OML models and provides an OML text editor that uses the language server protocol (LSP).

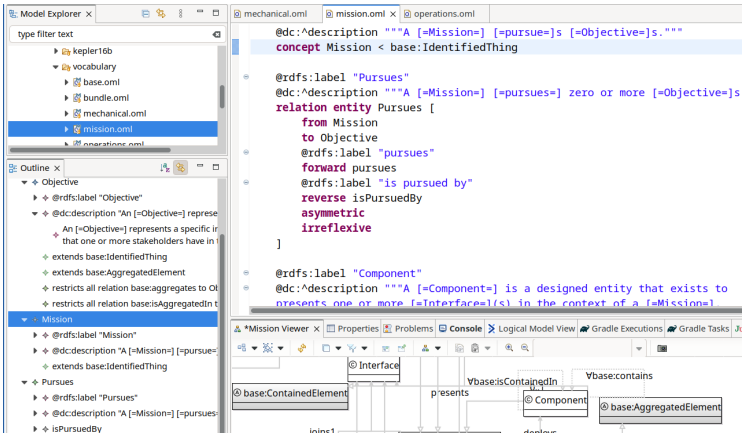

Fig. 8. Rosetta editor for creating ontological DSMLs.

OML models are text-based such that they can be source controlled using familiar file-based source control systems like Git. This enables familiar collaborative workflows involving commits, branches, and peer review before merging, for easy federation and integration of OML models.

The Rosetta IDE also leverages the EMF and Sirius to develop domain-specific viewpoints for OML descriptions. For instance, a table-based editor can be defined for the Kepler16b mission objectives. Using a domain-specific view can help create correct-by-construction OML descriptions and detect or prevent authoring errors. They can also address usability concerns, while retaining the rigor of OML description modeling under the hood. This was a key technique for the Europa Clipper wiring harness models, such that correctness and consistency were maintained through a DSML and a tool conforming to that DSML [48].

OML vocabularies thus provide a common DSML for the models of the systems. Analyses rely on these defined vocabularies to query the models and provide reports for system stakeholders such as management, subsystem engineers, and operators. Other modeling tools can also author OML description models by aligning with these vocabularies and using *OML adapters*. These tools convert back and forth between OML and other modeling formalisms. This includes an *owl-adapter* which currently supports OML to OWL, and an *ecore-adapter*, which currently supports Ecore to OML. Other adapters exist as prototypes (e.g., to UML/SysML) and some others (e.g., Modelica, SMT, Alloy) are planned for the future. This will allow OML to act as a common formalism to thread together engineering datasets and invoke existing analysis tools.

Analysis. openCAESAR provides Java-based analysis tools for OML that are packaged as task types for the Gradle build system and building on the adapters described above. These tools are configured in an OML project's `build.gradle` script, to be invoked from editor UIs or in a continuous integration (CI) pipeline.

The first set of tools are *OML tools* for OML models: validation, converting between serializations, and generating HTML documentation. The second set are *OWL tools* working on OWL models generated from OML: operating a Fuseki database, running a DL reasoner for logical consistency, executing SHACL constraint rules, running SPARQL queries for further checks and reporting, and generating OWL documentation.

Reporting. Non-technical stakeholders, such as JPL management, often require high-level reports about the system under development. openCAESAR reporting is also implemented through Gradle tasks. SPARQL queries are executed with the *owl-query task*, the reductions (processing the query results) are authored in a language (such as Javascript), and the rendering is performed with over-the-shelf frameworks such as D3 and Bikeshed. These reports can run automatically upon commits to the source control repository using CI scripts (e.g., Travis or Github Actions). The resulting artifacts can then be published to a web server (e.g. Github Pages) to make them accessible to stakeholders.

5 OML Adoption Examples

Europa Clipper electrical design One such successful application of OML and openCAESAR was in the Europa Clipper spacecraft project [48]. Here, a custom DSML and editor were created for system engineers to develop the spacecraft wiring harness. Rather than the existing error-prone approach based on spreadsheets, the custom editor utilized OML vocabularies as a DSML. This editor could then provide semantic error and integrity checks, catching or preventing errors entirely and reducing the time taken for modeling. For example, referential errors of connecting electrical components were eliminated using this approach, as the editor only exposed relevant components. This efficiency gain, while maintaining a familiar spreadsheet-like interface, led the team to switch from the previous spreadsheet-based approach in about six weeks. Forty reporting templates were created to expose different views on the system model (electrical, architecture, design, etc.), made consistent by aligning to these vocabularies.

Satellite power analysis At the Japan Aerospace Exploration Agency (JAXA), OML and openCAESAR was used to enable a short turn-around power analysis for a Very Low Earth Orbit (VLEO) satellite [35]. This power analysis studies operational scenarios and system configurations to determine the trade-off between satellite power consumption and solar panel/battery operation. This planning involves when to observe, send data, fire thrusters, and heat components.

The previous version of this analysis involved manual information transfer between domain engineers and systems engineers with spreadsheets and presentation slides. Tedious manual data entry was used to transfer data between three separate analysis tools for orbit analysis, operation scenarios, and power balance.

With the openCAESAR and OML-based approach, a workflow orchestration platform automated this analysis. OML vocabularies were used to represent concepts such as *work package, component, mass properties, and power modes*, and

analysis patterns such as *mass and power aggregation* [24]. An OML Metrology Vocabulary was also used [42], building on the International Vocabulary of Metrology (VIM) to represent measurements, quantities, and units.

Here, the original spreadsheets were kept as the system engineer input interface. When updated, the CI/CD analysis workflow transformed the spreadsheets into OML Descriptions using the R language, performed model queries, reasoning, and simulation, and produced reports through SPARQL queries. The outcome is that manual bottlenecks and errors were reduced, shortening analysis timeline from weeks to hours. This means that continual analysis can be performed over eight orbital periods, an improvement over the manual approach.

Another JAXA project enabled ontological reasoning for SysML models [36]. Here, an OML vocabulary represented the SysML meta-model. JAXA SysML models were then transformed into OML descriptions conforming to this vocabulary. With the addition of constraints and inference rules in the vocabulary, the description logic reasoners detected three consistency error patterns. SPARQL queries were also used to validate the model, providing greater reusability and extensibility over SysML-based checks. One insight reported here is that using inference rules in the vocabulary led to more concise validation queries, demonstrating the utility of logical reasoning in systems engineering.

OML as a SE Methodology (Leonardo UK) Humphries *et al.* propose the Integrated Knowledge-Centric Engineering (IKCE) methodology, providing a flexible and automated framework to increase avionic project velocity while maintaining rigor [23]. OML vocabularies are the common ontology to integrate and bridge data formats and semantics. For example, these vocabularies provide concepts for requirements and operational analyses, system architecture, and project management. Data is transformed from engineering tools and repositories into a common RDF representation, with a discovery portal available to aid reusability.

One innovation of the IKCE methodology is its emphasis on federation. The concept of *workspaces* divides up the project into concurrent development units. OML description models are versioned, released, and pushed/pulled between workspaces with a package registry system. This allows for both precise control of models and data by teams, but also distributed model development while maintaining semantic consistency.

Academic Uptake (Belgium, Canada, USA) Mittal *et al.* utilize OML to describe *validity frames* for models [32]. These validity frames capture the conditions (e.g. temperature range) under which a model is said to be *valid* [45], which is important for *design-space exploration* during the systems engineering process. In related work, Rys *et al.* use OML to define vocabularies for SE workflows and model versioning information [43]. They then provide a model repository/knowledge graph, an editor for SE workflows, and a workflow enactment engine for performing model management during workflow execution, improving traceability and reducing user error. Oakes *et al.* propose an ontology-based methodology for defining digital twin (DT) services [37]. OML vocabularies relate the structure of the DT with modeled workflows that a user follows to define, connect,

and deploy the DT components. Follow-up work by Fiter *et al.* provides a tool to query the DT model and visualize the DT's conceptual architecture [11] by incorporating a DT Description Framework [12]. Gregory *et al.* uses OML as part of a Digital Engineering Factory to support engineering students [14].

Lessons Learned These applications of OML/openCAESAR in practice demonstrate three lessons: 1) *Semantic technologies are a hidden backbone to the projects.* Familiar interfaces are kept and system engineers do not interact directly with the ontologies, yet the consistency and interoperability benefits remain. However, a methodologist must still maintain ontologies and tooling. 2) *System engineering is made more robust and efficient with ontological techniques.* The spacecraft applications demonstrate that OML and its methodologies provide agile yet rigorous foundations to reduce development and analysis time (weeks to hours), through automation potential and enhanced error detection or prevention (e.g., referential errors, consistency violations). Non-trivial toolchains may need to be constructed and optimized to support this. 3) *Best practices from systems engineering, semantic web technologies, and software engineering must be integrated*, including ontology reuse across (sub-)projects, automated analysis and reporting, explicit versioning for federation, and Git best practices (e.g. commits, branching). This may require extensive user training (i.e. Git commands).

6 Conclusion

This article describes how the OML language, ecosystem, and methodology is used in practice to bridge semantic technologies and model-based systems engineering. Both disciplines are facing related knowledge modeling challenges, and OML and openCAESAR offer an agile yet rigorous solution to address them, as seen from research groups, industry, and governmental institution uptake.

Both NASA JPL and JAXA are committed to governance and management of OML and openCAESAR, including feature development, maintenance, documentation production, and continued open-sourcing of internal tools where possible. Object Management Group (OMG) discussions are ongoing on how to integrate OML and openCAESAR with SysML. The community surrounding OML and openCAESAR is active, with annual events and online groups for academics and industrial partners: `https://www.opencaesar.io/events/`.

We hope that this article demonstrates the challenges and benefits of applying semantic web technologies to systems engineering. We invite the semantic web community to lend their expertise to help develop solutions for the ever-common federation, consistency, and integration problems encountered in systems engineering. One possible direction would be to extend OML to handle full OWL-DL without adding undesired complexity for systems engineers. We also see a challenge in extracting domain knowledge from systems engineering artefacts with implicit meta-models, which require domain expertise to elicit [36].

Supplemental Material Statement All OML resources are available: `https://www.opencaesar.io/oml/` and `https://www.opencaesar.io/oml-tutorials/`

Declaration of use of Generative AI Generative AI was used only to identify minor editing improvements, such as clarifications and consistency issues. No rewriting was performed with generative AI.

References

1. Antoniazzi, F., Viola, F.: RDF graph visualization tools: a survey. In: 23rd Conference of Open Innovations Association (FRUCT) (2018). https://doi.org/10.23919/FRUCT.2018.8588069
2. Arp, R., Smith, B., Spear, A.D.: Building ontologies with basic formal ontology. Mit Press (2015)
3. Bajaj, M., Friedenthal, S., Seidewitz, E.: Systems modeling language (SysML v2) support for digital engineering. Insight **25**(1), 19–24 (2022). https://doi.org/10.1002/inst.12367
4. Bracha, G., Cook, W.: Mixin-based inheritance. In: European Conference on Object-Oriented Programming on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA/ECOOP). ACM (1990). https://doi.org/10.1145/97945.97982
5. Chao, L.P., Tumer, I., Ishii, K.: Design process error-proofing: Engineering peer review lessons from NASA. In: International Design Engineering Technical Conferences and Computers and Information in Engineering Conference. vol. Volume 3d: 8th Design for Manufacturing Conference. ASME (2004). https://doi.org/10.1115/DETC2004-57764
6. Drewien, C.A., Artery, G., Eras, K., Ballard, W., Nagel, J.F.: Effective system engineering peer reviews. In: INCOSE International Symposium. vol. 25. Wiley Online Library (2015). https://doi.org/10.1002/j.2334-5837.2015.00093.x
7. Eclipse Foundation: Eclipse sirius. https://www.eclipse.org/sirius/ (nd), accessed: 2026-04-14
8. Elaasar, M., Hamou-Lhadj, A., Oakes, B., Hamdaqa, M.: Model-based systems engineering perspectives: A survey of practitioner experiences and challenges. In: Proceedings of the System Analysis and Modelling conference (SAM). ACM (2025)
9. Elaasar, M., Rouquette, N., Wagner, D., Oakes, B., Hamou-Lhadj, A., Hamdaqa, M.: openCAESAR: Balancing agility and rigor in model-based systems engineering. 2023 International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C) (2023). https://doi.org/10.1109/MODELS-C59198.2023.00051
10. Eysholdt, M., Behrens, H.: Xtext: implement your language faster than the quick and dirty way. In: SPLASH/OOPSLA Companion. pp. 307–309. ACM (2010). https://doi.org/10.1145/1869542.1869625
11. Fiter, K., Malassigné-Onfroy, L., Oakes, B.: DTInsight: A tool for explicit, interactive, and continuous digital twin reporting. In: Proceedings of the 28th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings. MODELS Companion '25 (2025)
12. Gil, S., Oakes, B., Gomes, C., Frasheri, M., Larsen, P.G.: Towards a systematic reporting framework for digital twins: A cooperative robotics case study. SIMULATION pp. 1–27 (2024). https://doi.org/10.1177/00375497241261406
13. Golbreich, C., Wallace, E.K.: OWL 2 Web Ontology Language New Features and Rationale (Second Edition). W3c recommendation, World Wide Web Consortium (W3C) (Dec 2012), https://www.w3.org/TR/owl2-new-features/, section 4.1: Syntax

14. Gregory, J., Salado, A., O'Neal, S., Larez, R., Reda, C., Martell, N., Martin, E., Colson, M., Masterson, J., Armenta, D.: The digital engineering factory: Considerations, current status, and lessons learned. In: INCOSE International Symposium. vol. 34, pp. 927–943. Wiley Online Library (2024)
15. Guizzardi, G., Wagner, G., Almeida, J.P.A., Guizzardi, R.S.: Towards ontological foundations for conceptual modeling: The Unified Foundational Ontology (UFO) story. Applied ontology **10**(3-4) (2015). https://doi.org/10.3233/AO-150157
16. Haase, P., Lewen, H., Studer, R., Tran, D.T., Erdmann, M., d'Aquin, M., Motta, E.: The NeOn ontology engineering toolkit. WWW (2008)
17. Henderson, K., Salado, A.: Value and benefits of model-based systems engineering (MBSE): Evidence from the literature. Systems Engineering **24**(1) (2021). https://doi.org/10.1002/sys.21566
18. Henderson-Sellers, B.: Bridging metamodels and ontologies in software engineering. Journal of Systems and Software **84**(2), 301–313 (2011). https://doi.org/10.1016/j.jss.2010.10.025
19. Hennig, C., Viehl, A., Kämpgen, B., Eisenmann, H.: Ontology-based design of space systems. In: International Semantic Web Conference. pp. 308–324. Springer (2016). https://doi.org/10.1007/978-3-319-46547-0_29
20. Hildebrandt, C., Köcher, A., Küstner, C., López-Enríquez, C.M., Müller, A.W., Caesar, B., Gundlach, C.S., Fay, A.: Ontology building for cyber–physical systems: Application in the manufacturing domain. IEEE Transactions on Automation Science and Engineering **17**(3), 1266–1282 (2020). https://doi.org/10.1109/TASE.2020.2991777
21. Hitzler, P.: A review of the semantic web field. Commun. ACM **64**(2), 76–83 (2021). https://doi.org/10.1145/3397512
22. Horridge, M., Drummond, N., Goodwin, J., Rector, A.L., Stevens, R., Wang, H.: The Manchester OWL syntax. In: OWLED. CEUR Workshop Proceedings, vol. 216. CEUR-WS.org (2006)
23. Humphries, L., Bullegas, G., Golledge, J., Mccausland, C., Taylor, D., Kolovos, D., Garcia-Dominguez, A., Gerasimou, S.: Integrated Knowledge Centric Engineering: Delivering next-generation aircraft projects at pace . In: 2025 ACM/IEEE 28th International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C). pp. 834–843. IEEE Computer Society, Los Alamitos, CA, USA (Oct 2025). https://doi.org/10.1109/MODELS-C68889.2025.00110
24. Jenkins, S., Rouquette, N., Elaasar, M.: IMCE vocabularies. https://www.opencaesar.io/imce-vocabularies/, online
25. Kamburjan, E., Cederbladh, J.: Explaining model-based systems engineering – towards a semiotic perspective. In: 35th INCOSE International Symposium. vol. 35, pp. 337–347 (2025). https://doi.org/10.1002/iis2.70070
26. Kamburjan, E., Pernisch, R., Corcho, Ó., Chaves-Fraga, D.: On dependencies in knowledge graph construction. In: KGCW@ESWC. CEUR Workshop Proceedings, vol. 3999. CEUR-WS.org (2025)
27. Kärnä, J., Tolvanen, J.P., Kelly, S.: Evaluating the use of domain-specific modeling in practice. In: Proceedings of the Object-Oriented Programming, Systems, Languages and Applications workshop on Domain-Specific Modeling (2009)
28. Kelly, S., Tolvanen, J.P.: Domain-specific modeling: enabling full code generation. John Wiley & Sons (2008)
29. Khan, Z.C., Keet, C.M.: Dependencies between modularity metrics towards improved modules. In: EKAW. Lecture Notes in Computer Science, vol. 10024, pp. 400–415 (2016). https://doi.org/10.1007/978-3-319-49004-5_26

30. Lohmann, S., Negru, S., Haag, F., Ertl, T.: Visualizing ontologies with VOWL. Semantic Web **7**(4), 399–419 (2016). https://doi.org/10.3233/SW-150200
31. Madni, A.M., Sievers, M.: Model-based systems engineering: Motivation, current status, and research opportunities. Systems Engineering **21**(3) (2018). https://doi.org/10.1002/sys.21438
32. Mittal, R., Eslampanah, R., Lima, L., Vangheluwe, H., Blouin, D.: Towards an ontological framework for validity frames. In: International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C). pp. 801–805. IEEE (2023). https://doi.org/10.1109/MODELS-C59198.2023.00128
33. Moon, D.A.: Object-oriented programming with flavors. In: OOPSLA. pp. 1–8. ACM (1986). https://doi.org/10.1145/960112.28698
34. Musen, M.A., Protégé Team: The protégé project: A look back and a look forward. AI Matters **1**(4), 4–12 (2015). https://doi.org/10.1145/2757001.2757003
35. Nakajima, Y., Katsumata, H., Tate, D., Komatsu, Y., Levitt, A., Jenkins, J.S., Elaasar, M., Rouquette, N.F., Wagner, D.A.: openCAESAR Application to Power Balance Analysis in Early Space Mission Formulation . In: 2025 ACM/IEEE 28th International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C). pp. 844–852. IEEE Computer Society, Los Alamitos, CA, USA (Oct 2025). https://doi.org/10.1109/MODELS-C68889.2025.00111
36. Nakajima, Y., Wada, A., Komatsu, Y., Elaasar, M., Jenkins, J.S., Wagner, D.A.: Power of a Reasoner: Model Validation for SysML Model using openCAESAR . In: 2025 ACM/IEEE 28th International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C). pp. 853–857. IEEE Computer Society, Los Alamitos, CA, USA (Oct 2025). https://doi.org/10.1109/MODELS-C68889.2025.00112
37. Oakes, B., Gomes, C., Kamburjan, E., Abbiati, G., Ecem Bas, E., Engelsgaard, S.: Towards ontological service-driven engineering of digital twins. In: Proceedings of the ACM/IEEE 27th International Conference on Model Driven Engineering Languages and Systems. pp. 464–469 (2024). https://doi.org/10.1145/3652620.3688261
38. openCAESAR: OML Luxor. https://github.com/opencaesar/oml-luxor (nd), GitHub repository, Accessed: 2026-04-14
39. Parrott, E.L., Weiland, K.J.: Using model-based systems engineering to provide artifacts for NASA project life-cycle and technical reviews. In: AIAA Space Conference. AIAA (2017). https://doi.org/10.2514/6.2017-5299
40. Qu, Y., Kamburjan, E., Skjæveland, M.G., Waaler, A.: Ontology alignment and system models. In: INCOSE International Symposium (2026), accepted for publication, preprint available under edkamb.github.io/files/incose26.pdf
41. Reynolds, O.J., Garcia-Dominguez, A., Kolovos, D., Bullegas, G., Mccausland, C.: Towards bridging ontological and closed-world modelling with synchronised EMF views of RDF models . In: 2025 ACM/IEEE 28th International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C). pp. 858–867. IEEE Computer Society, Los Alamitos, CA, USA (Oct 2025). https://doi.org/10.1109/MODELS-C68889.2025.00113
42. Rouquette, N., Elaasar, M.: Metrology vocabularies. https://www.opencaesar.io/metrology-vocabularies/, online
43. Ryś, A., Lima, L., Exelmans, J., Janssens, D., Vangheluwe, H.: Model management to support systems engineering workflows using ontology-based knowledge graphs. Journal of Industrial Information Integration **42** (2024). https://doi.org/10.1016/j.jii.2024.100720

44. Singam, C., Carter, J.: Model-based systems engineering (mbse). In: Hutchison, N. (ed.) The Guide to the Systems Engineering Body of Knowledge (SEBoK). The Trustees of the Stevens Institute of Technology, Hoboken, NJ, v. 2.12 edn. (2024), https://www.sebokwiki.org
45. Van Mierlo, S., Oakes, B., Van Acker, B., Eslampanah, R., Denil, J., Vangheluwe, H.: Exploring validity frames in practice. In: International Conference on Systems Modelling and Management. pp. 131–148. Springer (2020). https://doi.org/10.1007/978-3-030-58167-1_10
46. Voelter, M., Kolb, B., Birken, K., Tomassetti, F., Alff, P., Wiart, L., Wortmann, A., Nordmann, A.: Using language workbenches and domain-specific languages for safety-critical software development. Software & Systems Modeling **18**(4) (2019). https://doi.org/10.1007/s10270-018-0679-0
47. Wagner, D.A., Chodas, M., Elaasar, M., Jenkins, J.S., Rouquette, N.: Ontological Metamodeling and Analysis Using openCAESAR. In: Madni, A.M., Augustine, N., Sievers, M. (eds.) Handbook of Model-Based Systems Engineering, pp. 1–30. Springer International Publishing, Cham (2022). https://doi.org/10.1007/978-3-030-27486-3_78-1
48. Wagner, D., Kim-Castet, S.Y., Jimenez, A., Elaasar, M., Rouquette, N., Jenkins, S.: CAESAR model-based approach to harness design. In: IEEE Aerospace Conference. IEEE (2020). https://doi.org/10.1109/AERO47225.2020.9172630
49. Walden, D.D., Shortell, T., Roedler, G., Delicado, B., Mornas, O., Yew-Seng, Y., Endler, D.: INCOSE Systems Engineering Handbook, 5th Edition. Wiley (2023), https://www.incose.org/publications/se-handbook-v5
50. Walter, T., Parreiras, F.S., Staab, S.: OntoDSL: An ontology-based framework for domain-specific languages. In: Model Driven Engineering Languages and Systems: 12th International Conference, MODELS 2009, Denver, CO, USA, October 4-9, 2009. Proceedings 12. pp. 408–422. Springer (2009). https://doi.org/10.1007/978-3-642-04425-0_32
51. Walter, T., Parreiras, F.S., Staab, S.: An ontology-based framework for domain-specific modeling. Software & Systems Modeling **13**(1), 83–108 (2014). https://doi.org/10.1007/s10270-012-0249-9
52. Wasowski, A., Berger, T.: Domain-Specific Languages - Effective Modeling, Automation, and Reuse. Springer (2023). https://doi.org/10.1007/978-3-031-23669-3, https://doi.org/10.1007/978-3-031-23669-3
53. Wirth, N.: The module: A system structuring facility in high-level programming languages. In: Language Design and Programming Methodology. Lecture Notes in Computer Science, vol. 79, pp. 1–24. Springer (1979). https://doi.org/10.1007/3-540-09745-7_1
54. Yang, L., Cormican, K., Yu, M.: Ontology-based systems engineering: A state-of-the-art review. Computers in Industry **111**, 148–171 (2019). https://doi.org/10.1016/j.compind.2019.05.003