Augmenting Model-Based Systems Engineering with Knowledge

Luis Palacios Medinacelli
luis.palacios@cea.fr
Université Paris-Saclay
CEA, List
Palaiseau, France

Florian Noyrit
florian.noyrit@cea.fr
Université Paris-Saclay
CEA, List
Palaiseau, France

Chokri Mraidha
Chokri.Mraidha@cea.fr
Université Paris-Saclay
CEA, List
Palaiseau, France

ABSTRACT

This article presents a general approach for the integration of Knowledge Bases into Model-Based Systems Engineering tools. In existing tools, domain-specific modeling languages are well supported. However when it comes to enforcing design constraints, existing approaches are verbose, it is difficult to be complete and consistent, and the reuse of knowledge is only possible in a limited way (mainly through model libraries). Furthermore, current tools usually lack or have limited capability to detect semantic errors, ability to evaluate the models with respect to formal expert knowledge, and the ability to understand what is being designed. Our work addresses these limitations through the semantic annotation of UML models in Papyrus (an MBSE Tool), to attach domain-specific semantics to the models. This integration enables not only reasoning capabilities over the annotated models, but the models can be shared with semantic-compatible tools and stakeholders. Moreover, the models can reuse and integrate knowledge generated outside the tooling environment. The approach's feasibility is demonstrated through an implementation that defines a technology stack, with emphasis on the mapping of UML elements and its counterparts in the ontology. We address the coherence and preservation of the semantics throughout the transformation process, which enable the formalization of constraints coming from the UML's system design. Finally, we illustrate the reasoning capabilities by evaluating expert knowledge via SPARQL queries and SWRL rules.

CCS CONCEPTS

• **Theory of computation** → **Semantics and reasoning**; • **Computer systems organization** → *Robotics*.

KEYWORDS

Model-Driven Engineering, Knowledge Based Engineering, Semantic Interoperability, Ontology, UML, Papyrus

ACM Reference Format:
Luis Palacios Medinacelli, Florian Noyrit, and Chokri Mraidha . 2022. Augmenting Model-Based Systems Engineering with Knowledge. In *ACM/IEEE 25th International Conference on Model Driven Engineering Languages and Systems (MODELS '22 Companion), October 23–28, 2022, Montreal, QC, Canada.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3550356.3561548

MODELS '22 Companion, October 23–28, 2022, Montreal, QC, Canada

ACM ISBN 978-1-4503-9467-3/22/10…$15.00
https://doi.org/10.1145/3550356.3561548

1 INTRODUCTION

Complex systems design, like Autonomous or Cyber-Physical Systems, require specific design and engineering techniques to ensure the resulting systems comply with its specifications. Among these techniques Model-Based System Engineering (MBSE) provides good practices and formalized syntax that make the engineering process systematic. It notably helps in sharing the same interpretation of the models among experts. Among the existing modeling languages available for complex system design we consider the largely adopted UML[1] and SysML[2]. The expressiveness of these languages is high, and thus current modeling tools are sophisticated with support for rich expressiveness to describe the systems being modeled, but at the expense of requiring high expertise in the tool's specific representation language. Moreover new projects have to be built from scratch, rebuilding structures and descriptions that are common to specific domains (e.g. Autonomous Systems), where knowledge reuse is available in a limited way (e.g. model libraries). Despite the formalized nature of MBSE, current tooling environments have limited interpretation capabilities regarding the content and semantics of the system being modeled.

In the domain of Artificial Intelligence (AI), the field of Knowledge Representation (KR) aims to formally structure knowledge, addressing its syntax, semantics and reasoning capabilities. Within the resources for KR, ontologies are widely used in AI as knowledge bases, thanks to their formal semantics, their tractable computational features and their solid mathematical background. The current W3C recommendation for ontologies syntax is OWL (Web Ontology Language). OWL shares the expressivity power and reasoning complexity features of Description Logics [3], providing OWL with a solid formal background and efficient reasoning algorithms.

There exists extensive research [10] that address the formal description of (semi) autonomous systems using ontologies, in applications like: human-machine & machine-machine interaction, robotic cell reconfiguration [16], formalization of skills and high level cognitive tasks for robots [4, 26], etc. These works emphasize several benefits that ontologies can bring to robotic systems: i) correct by design models, ii) sharing and reuse of knowledge, iii) advanced inferences involving complex definitions and high-level abstraction concepts iv) logic rules to further specify the design of a system, v) queries to identify patterns and anti-patterns, and vi) the reduction of heterogeneity in large systems.

To overcome the limits of MBSE tooling, *we aim to integrate intelligence coming from the formalized knowledge captured as ontologies*, thus enabling the aforementioned benefits. To bridge the

[1] https://www.uml.org/
[2] https://www.sysml.org/

gap between MBSE tooling and KR, we have identified the following challenges:

- The knowledge captured in the ontology should be automatically made available to the designer, saving time and avoiding mistakes.
- The system's model designed in the MBSE tooling environment should be described in terms of this existing knowledge. In our context this means the annotation of the UML model with the ontology concepts.
- The system's design should be exported from the MBSE environment as an OWL compliant representation, thus providing a tool-agnostic formal representation of the system.
- The enhanced models obtained this way need to be suitable for automatic reasoning tasks, like consistency and instance checking.

We argue that a solution to overcome these challenges relies on the ability to integrate standardized domain specific ontologies into the tooling environment, while ensuring the coherence of the transformations between formalisms. Under this assumption, our work addresses the feasibility of the integration of existing KBs and reasoning capabilities into MBSE tooling. In this context, we present two contributions: first, we develop an ontology for Unmanned Aereal Vehicles (UAVs) and integrate it into the IEEE standard ontology CORA [3]; and second, we provide an approach and a technology stack for the integration and export of the formalized knowledge from MBSE tooling environment, which is implemented using the ontology for UAVs and the Papyrus[4] tool.

The rest of this article is organized as follows: in section 2 we present the works related to ontologies and MBSE. In section 3 we present our approach, along with some preliminaries. In section 4 we present an implementation of the approach, motivated by the drone use case. Finally, we present our conclusions & further works in section 5.

2 BACKGROUND & RELATED WORKS

In this section we present some of the most relevant works regarding ontologies, UML and MBSE. These works present the motivations, sate of the art, challenges and envisaged benefits from the interaction of the aforementioned topics.

2.1 Ontologies and Autonomous Systems

A central reference for our work is the Core Ontology for Robotics and Automation (CORA) [11, 24] published in 2015, the first-ever standard from the IEEE Robotics and Automation Society [15] . The CORA ontology uses SUMO as its top-level ontology, which defines basic ontological categories across all domains. This standard provides formalized OWL versions of the ontology for free access [25], modularized in different levels of axiomatization and scope: CORAX (high level general concepts), CORA (where *robot* is the main concept), RPARTS (defines robot parts as roles) and POS (an ontology about position, orientation and pose). The ontologies provide a consistent set of the core concepts relevant to the domain of robotics, yet remaining relevant for general purpose.

Regarding KR based systems and ontologies for autonomous systems, we refer to two main surveys: [22] from 2019, and [10] from 2021. From the explored approaches in these surveys, we highlight the following as the most relevant works related to our research:

KnowRob[5] is an open source knowledge processing system that is designed for autonomous service robots [26]. Its purpose is to equip robots with the capability to organize information in re-usable knowledge chunks, and to perform reasoning in an expressive logic. It further provides a set of tools for visualization and acquisition of knowledge.

Perception and Manipulation Knowledge (PMK) [4] is a knowledge-based reasoning framework that includes some reasoning processes for autonomous robots to enhance Task and Motion Planning (TAMP) capabilities in the manipulation domain. The reasoning scope of PMK is divided into four parts: i) reasoning for perception, ii) reasoning for object features, iii) reasoning for a situation, and iv) reasoning for planning.

Regarding the specific field of UAV's, the dronetology[6] ontology considers the main elements to describe the physical components of drones, as well as missions, flight paths and information about weather. This ontology uses the IEEE sensors ontology SSN [6] , the ontology for spatial positions and relations and the Weather Ontology to complete its model. Recent work [17] uses this approach for drone navigation in the *dynamic sensor harvesting* use case. Some limits of this ontology are the lack of documentation on the definitions and the application-specific target.

The Manufacturing Resource Capability Ontology (MaRCO) [16] is an OWL-based ontology developed to describe the capabilities of manufacturing resources. MaRCO supports the representation and automatic inference of combined capabilities from the representation of the simple capabilities of co-operating resources. Their representations are modularized in four main models : i) *Product*, ii) *Process*, iii) *Capability* and iv) *Resources*.

2.2 The UML and Ontologies Formalisms

Both, the UML and Ontologies formalisms, allow to describe a domain of interest in terms of concepts/classes, objects of these classes, and their relations. Both can describe knowledge in a two-layer structure: a first layer defines the abstract structure of the domain of applications, and the second layer is the instantiation of the first one. Yet some fundamental differences exist [20]. Notably, ontologies make the Open World Assumption (OWA), while UML makes the Closed World Assumption (CWA), meaning that all the relevant information is available in the model. Another difference relies on the relations, which in the UML exist only between two or more classes, whereas in OWL they can be defined independently between objects.

A recurrent issue addressed by works combining UML and ontologies is the problem of semantic heterogeneity in distributed and delocalized companies , where problems of misunderstanding and information exchange may arise, due to different viewpoints, for which applications are developed [8, 23]. There is also the risk of loss of information when exchanging between heterogeneous

[3] https://github.com/srfiorini/IEEE1872-owl, https://ieeexplore.ieee.org/document/7084073
[4] https://www.eclipse.org/papyrus/
[5] http://knowrob.org/
[6] http://www.dronetology.net/

systems. In these works, the use of ontologies as models is proposed, to trace relevant and shared information related to the knowledge domain in question.

There are also some problems, reported by ODM (Ontology Definition Metamodel), in metamodel transformation from UML to OWL [9]. The issue of *structure conflation* occurs when two constructs in the source metamodel map to a single construct in the target metamodel. There is also the problem of *structure loss* when a complex construct is transformed into a collection of simpler constructs, and the problem of lack of constructs on the target metamodel that are available in the source metamodel.

The work in [1] evidences that there is a lack of a complete mapping between the constructs of the two languages, but there might be specific concepts and relations in the ontology, as in the case of a Domain Specific Language (DSL), for which some UML constructs are suitable.

From these works, the authors in [20] conclude that: a) there is an incomplete syntactic correspondence between UML class diagram constructs and OWL, b) other diagrams are often excluded from the transformations, neglecting essential knowledge existing in ignored diagrams, and that c) there is rarely an evaluation of the resulting ontology, that is, information about the computational efficiency of the ontology and its ability to fulfill the required tasks, such as query answering, classification or consistency checking.

2.3 Ontologies and MBSE

According to the literature [29] there are several areas of system engineering (SE) knowledge where research between ontologies and SE has been conducted: *System Fundamentals*, *System engineering Standards*, *Generic LifeCycle Stages*, *Representing Systems with Models*, *Engineered System Contexts* and *System Engineering Management*. The works considered in [29] summarize the causes for the difficulties in developing systems on budget and on time, and the considerable resource waste dealing with the correction of mistakes, into four reasons: 1) the implicit nature of SE, 2) the limitations of best-practice standards and meta-models, 3) the absence of a widely accepted and consistent terminology, and 4) inefficient collaborations due to the misunderstanding and misinterpretation.

The rationale of using ontologies in SE aims to mitigate some of these problems: although there exists standards on how to implement best practice of SE processes, like ISO/IEC/IEEE 15288, they are not computer interpretable [28]. On the side of the MBSE approach, existing meta-models use a language which is sometimes unfamiliar to some intended users [13]. Moreover, different interpretations by individuals and communities can lead to the misunderstanding and misinterpretation in the development of systems [5, 14]. Research has shown that significant system failure costs are due to a lack of adequate information exchange and communication within projects, inconsistent technology adoption among stakeholders, and continued paper-based business practices [27].

Ontologies can improve system design, by facilitating communication among stakeholders having different concerns when designing, for example, a Cyber-Physical-System. Common, interdependent properties can be harmonized and synchronized, to manage inconsistencies [12]. Thus ontologies can ensure that multiple systems share a common terminology, which is the essence of knowledge sharing and reuse. Formal definitions for the different properties and processes of SE would be a significant contribution towards improving accuracy and precision in the implementation of SE [19]. And, by using a predefined ontology, it is possible to reduce the number of misinterpretations within projects [14].

3 APPROACH

Our approach considers the problem of standardized domain specific ontologies and their integration into MBSE tools. Three main aspects of knowledge management [18, 21] are addressed in this problem: First, we require this integration to enable *expert knowledge reuse*. That is, the ability to evaluate some aspects of our design with respect to expert's knowledge constraints expressed in the ontology. Then, this integration would enable *knowledge acquisition*, thanks to the formal representation of the expert's system's design, as OWL. Finally, the re-use of models and their further specialization in the MBSE tool, contributes to *knowledge refinement*.

To achieve these objectives, we resource to semantic technologies which provide the means to formally represent domain knowledge, and perform reasoning operations over it. Next, we target a MBSE tooling environment, i.e. Papyrus, for which a series of resources (mappings, formalisms and adapters) need to be defined and organized, to enable the integration of the domain knowledge. Out of this integration, the annotated system model still needs to be exported as an OWL representation to perform useful inferences like consistency, compliance and instance checking.

Thus we have divided the resources and results of our approach accordingly. In the following sections, we present the Semantic Technologies, and Mapping & Adapters used in our approach.

3.1 Semantic Technologies

The vocabularies used in our implementation have several origins and purposes. At the top level we have the Core Ontology for Robots and Autonomous Systems (CORA – IEE1872) which is an IEEE standard, and is itself composed of several ontologies/modules. This ontology serves as a "context" for integrating the domain specific vocabularies (i.e. a specific vocabulary about a type of autonomous system), since the level of abstraction of CORA is too general to be directly used in a specific application domain. A Drone Components ontology, *ODrone*, comprises the most common types of devices used in a drone, their classification, and some of their most common relations (connected to, energy supply, etc), thus tackling specific viewpoints and concerns. This ontology has been developed in the context of the Cyber-Physical Systems for European Union project[7] (CPS4EU).

It is crucial that an ontology that is intended to be used among organizations describes its captured knowledge in a shared and standardized vocabulary and semantics [20]. To this end, we have integrated *ODrone* into CORA, specifically targeting two aspects: the definition of the drone and its parts, and the role these parts play in the system. Accordingly, the core concepts of CORA (*Robot*, *Device*) and the relations defined in RPARTS[8] constitute the upper-level concepts and relations of *ODrone*.

[7] https://cps4eu.eu/
[8] A sub-ontology from CORA.

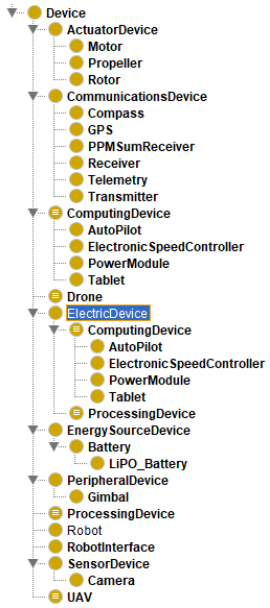

Figure 1: The main concepts of *ODrone* and their integration into CORA.

3.1.1 ODrone. The purpose of the *ODrone* ontology is to provide a core vocabulary for describing drones and the parts that compose it, i.e. the devices and components that build up a drone. This vocabulary enables *System Designers* to describe their systems in terms of the ontology. The *ODrone* ontology can be directly used or further specialized to consider specific types of devices (e.g. a special type of battery or motor) while preserving the standardized vocabulary and the semantics of the upper ontologies. An additional set of ontologies are used in the approach to describe quantities and units of measurement for describing the properties of the devices and systems in a standardized way.

***ODrone* Concepts:** The main concept of the *ODrone* ontology is *Device*. This is the main "insertion" point of the concepts of ODrone (fig. 1) into CORA, thus meaning that all newly defined sub-concepts inherit the properties of the CORA taxonomy (i.e. *Device isA Agent*, *Device isA Physical Entity*, etc.). Guided by the concepts in Cyber-Physical systems, five main types of devices are defined:

(1) *ActuatorDevice*
(2) *SensorDevice*
(3) *CommunicationsDevice*
(4) *ComputingDevice*
(5) *EnergySourceDevice*

In order to extend CORA by integrating the concepts in *ODrone*, we require the use of concepts and relations from more than one of CORA sub-ontologies. Figure 2 shows the integration of the *ActuatorDevice* concept from *ODrone* into CORA, as well as the involved ontologies. The integration of the other types of devices into CORA is analogous.

***ODrone* Object Properties:** The object properties in an ontology define the relations between its concepts and between their

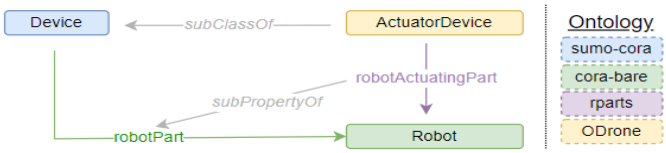

Figure 2: Integration of *ActuatorDevice* concept, from *ODrone*, into CORA. The concept *Device* is defined in sumo-cora, *Robot* and *robotPart* in cora-bare, and *robotActuatingPart* in rparts.

instances. Restricting the *domain* and the *range* of these relations, provides their formal semantics. These are specified next.

Regarding the *parts* that compose a system, two main relations from CORA are of interest: the *part* and the *robotPart* relations. *ODrone* provides five main types of devices, accordingly there are five main roles a device can play when composing a system :

(1) *actuatingPart*
(2) *communicatingPart*
(3) *energySourcePart*
(4) *processingPart*
(5) *sensingPart*

If these parts compose a *Robot*, they become *robotParts* of each of the the previously introduced types (fig 3). Note that, even though only *robotEnergySourcePart* has been added by *ODrone*, the domain and ranges of all *robotParts*, are specified only in *ODrone*. Thus rendering the semantics of these CORA relations more specific. This specification is realized by restricting the domain of those relations to the respective types of devices.

Connections between Parts: The devices that play the role of a part of a drone system, are specified using the relations described in the previous section. Following the definitions in CORA, an *ArtificialSystem*[9] is composed of parts that in more or less degree interact with each other . To model this interaction we provide the *isConnectedTo* relation, and three specializations: 1)*suppliesTo* (where the source device supplies energy to the target electric device), 2) *bus* (where the source device supplies data to the target computing device) and 3) *attachedTo* (where the two devices are physically connected)

Similarly to *robotPart* relations, the *isConnectedTo* relation and its specializations, are as well formalized in OWL, and integrated with CORA (Figure 3).

3.2 Mappings and Adapters

In this section we introduce the workflow detailing the resources and their interaction, to implement the approach. We also present the details on the mappings between *ODrone* and UML.

3.2.1 OML and Adapters. The Ontological Modeling Language (OML) specification defines a tool-neutral language that permits representation, exchange and analysis of information. OML is designed after the W3C standard Ontology Web Language 2 (OWL2) making it expressive, modular and extensible. OML is used as the representation language in the OpenCaesar[10] initiative. This project aims to enable rigorous system design thanks to the integration of

[9]The formal specification of *ArtificialSystem* is found in CORAX axioms
[10]https://github.com/opencaesar

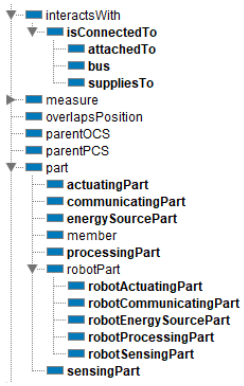

Figure 3: The main object properties in *ODrone*.

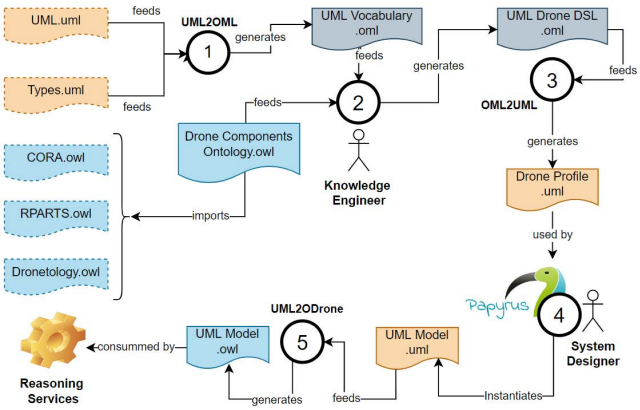

Figure 4: The workflow, resources and actors for the implementation of the approach. Tasks 1, 3 and 5 are automatic. Tasks 2 and 4 involve engineers and designers.

ontologies. Of special interest for our approach are the ease of use of OML which hides some of the complexity of OWL to define and manipulate ontologies, and the adapters which allow to transform OML descriptions into other formalisms.

3.2.2 Workflow. The process of integration of ontologies and a UML system model is illustrated in Figure 4 . The workflow is divided into five tasks, represented by the circles in the diagram, where dashed-border files represent external resources, and full-border files represent generated products of each process. File types are color-coded.

First, we have the UML specification[11] and its types, which are defined in .uml files (top-left of Figure 4). These files provide all the resources available in UML (classes, associations, etc.) that can be used to model a system.

Task 1 uses an UML to OML adapter to generate an UML vocabulary in OML format that comprises the specification and types of UML (part of the benefits of OML relies on the adapters available to transform and map different formalisms into OML).

[11]https://www.omg.org/spec/UML/2.0/

In task 2 we have the Domain Specific Language. This is the specific vocabulary used in the target domain of interest. Our work is encompassed within the autonomous systems domain, and specifically in UAV's subdomain. This language depends on, and is a subset of, the drone ontology *ODrone* described in the previous sections. The DSL needs to be specified in OML and its concepts mapped to UML. Specific *ODrone* concepts can be asserted as specializations of UML classes, and object and data properties can be asserted as specializations of UML associations. Specific semantics for these relations can also be defined in the OML mapping (domain, range, inverse relations, etc.) by a knowledge engineer.

Once the DSL.oml is ready, it feeds task 3, where the corresponding UML profile is automatically generated, thanks to the OML2UML adapter. This profile is an extension of UML, annotated by the relevant concepts in the *ODrone* ontology in task 2.

The profile can now be used to model a system in task 4, where it is imported into Papyrus UI and used by a system designer to specify a system whose elements carry the semantics of the ontology.

Finally, in task 5, the annotated UML model is transformed back to OWL, thanks to a custom UML2OWL adapter for the *ODrone* ontology, resulting in a W3C compliant representation of the system. This representation is suitable to be analyzed by reasoners. At this stage, constraints expressed as queries (SPARQL), rules (SWRL) or complex concept definitions (DL's/OWL) can be evaluated over the annotated UML model.

The previously described process enhances the UML model providing it with some added features: a) The design can be checked for consistency w.r.t. the semantics of the ontology, i.e. given a set of constraints in terms of the ontology the model can be checked for compliance. b) The UML design specifies a new type of system in terms of the concepts and relations in the ontology. This knowledge can be reused, and the specification becomes an OWL description that can effectively extends the ontology and restricts further models. c) Thanks to the attached semantics to the UML drone model, additional information coming from external sources can be integrated (e.g. a catalog of components can be used to instantiate the parts of the drone). d) The semantics of what is being modeled is available to the design application (Papyrus), this can be used to develop smart assistance for system designers.

3.2.3 Adapters & Mappings. Both, the UML and OWL languages, provide sets of constructors (classes, relations, individuals, etc.) that allow to describe real-world processes and entities in terms of the formalism. An UML or OWL model of a real world entity is an abstraction that considers only those elements relevant for the problem to solve. The intended meaning of the concepts and relations from the ontology, have to be preserved throughout the transformation into UML (as a profile) and back to OWL. Because of the variety of the constructors in both formalisms, their correlation is not only not evident, but case dependent. Moreover, there are specific sets of constructors and diagrams in UML for describing specific systems and entities (such as classes, relations, attributes, owned elements, composite structures and instance specifications).

To demonstrate the feasibility of the approach, we have implemented a mapping between *ODrone* and UML targeting: 1) Class Diagrams and 2) Composite Structure Diagrams. In the literature [20], it is common to target Class Diagrams constructs due to its

evident similarity with ontology constructs. Furthermore, Composite Structure diagrams, allow to further specify the relations of the parts of a system. Using these two diagrams the designer can describe the entities that play a role in his design and specific composite structures that carry interconnected parts.

Figure 5 summarizes the mapping between UML concepts and OWL/*ODrone* entities to ensure the ontology semantics are preserved. The mapping in figure 5 also enables the extension of the current ontology, by defining new classes in UML. These new classes, along with their definitions, become part of an extended ontology and can be reused by new designs, or exploited by external tools and services that are compatible with *ODrone* and CORA. Nevertheless, no mechanism for defining new relations via UML is provided at this stage, therefore new relations, their domain and range, must be already defined in the source ontology.

Constructors	UML (org.eclipse.uml2)	OWL (owlApi)
Concepts	uml:class	**owl:Class**
	stereotyped class	*class* **rdfs:subClassOf** *stereotype*
	class specialization	*newClass* **rdfs:subClassOf** *class*
	class generalisation	*class* **rdfs:subClassOf** newClass
	ownedElements (classifier)	**complex concept definition** (owl:equivalentClass)
Relations/Roles	stereotyped assotiation	**owl:ObjectProperty**
	uml:Assotiation, uml:Property	**owl:ObjectProperty**
	uml:ScalarProperty	**owl:ObjectDataProperty**
	sub properties	**rdfs:subPropertyOf**
	composite assotiation	**ODrone:part** (owl:ObjectProperty)
Individuals	InstanceSpecification	**owl:NamedIndividual**

Figure 5: Mapping between UML and *ODrone* constructors.

4 IMPLEMENTATION: DRONE USE CASE

As a proof of concept for our approach, we have developed an implementation of the integration of KBs into MBSE, to enhance and reason over UML models. The implementation applies the methodology, (namely the use of standardized ontologies, a domain specific ontology and the workflow) and defines a technology stack that provides the required functionalities. The next sections highlight the main features of the implementation.

4.1 Automatic Generation of the UML Profile

To generate the UML profile, we use as a pivot language OML [7] and rely on its Papyrus adapter. This adapter takes as input a set of vocabularies and a mapping from these vocabularies to the corresponding UML entities.

To start, a relevant subset of the vocabulary (*ODrone*) is selected, and expressed as OML. This defines the integration of the *ODrone* concepts and relations into the UML taxonomy. This OML description is then automatically transformed into an UML Profile, compatible with Papyrus. Note that depending on the use case and problem to solve, the profile can be extended or modified, while the process and principles remain the same.

4.2 Attach Semantics to an UML Design

Once the *ODrone* UML profile is available, we can use it to stereotype the UML elements of a system design. In Figure 6 on the left we have the class diagram of a *DroneSystem1* with the stereotype «Device» coming from *ODrone* (and CORA). On the right hand side of Figure 6, we can see the result of the transformation. In the ontology hierarchy, *DroneSystem1* is a sub-class of *Device*, which is its stereotype (left of Figure 6) and has *D_subSystem1* as a sub-class, as expected. The resulting hierarchy, is due to the mapping of the UML *stereotypes* and *generalization* relation into *ODrone*. Another, yet more elaborated, feature of the transformation, is the creation of ***complex concept definitions*** and its integration as *necessary and sufficient* conditions for the class being defined. In the example (fig. 6), the UML design specification of *DroneSystem1* requires that:

- The system is a *Device*,
- it is composed of *parts*: a *BatteryType1 part*, a *PropellerType1 part* and a *MotorType1 part*,
- the *BatteryType1 part suppliesTo* the *MotorType1 part*,
- the *MotorType isAttachedTo* the *PropellerType1 part*.

Accordingly we can see that the transformation process has created an equivalent class for *DroneSystem1* that considers and correctly formalizes these restrictions (right hand side of Figure 6). These complex concept definitions, which are an innovative achievement of our approach, can be checked for consistency at both: the specification level and the instances level. If inconsistencies are found, explanations on the sources of inconsistency can be obtained. This is an important aspect, since the formalization of restrictions, queries and rules on top of knowledge bases, can be tricky (in part due to the interpretation of formal semantics), may require high expertise, and can easily become complex. Our work allows the automatic generation of formal restrictions based on the targeted UML constructs.

4.3 Reasoning and Inferences

Given the transformation from the UML model design into OWL (thanks to the mapping), we can use a reasoner to draw inferences about the model. We introduce next a set of examples that illustrate basic inferences, the evaluation of automatically generated complex class definitions, and the reuse of expert knowledge coming from CORA.

Let us first introduce some preliminaries to properly present our examples. We adhere to the notation in [3], where predicates starting in uppercase, like *Device* or *SensorDevice*, denote concepts; predicates starting in lower case, like *sensingPart*, denote binary relations and subsumption is denoted by $C \sqsubseteq D$, meaning C is subsumed by D. Individuals are denoted by terms in lowercase and are asserted as instances of a class as parameters of unary predicates, like *Device(d1)*, or as part of a relation in binary predicates, like *hasPart(d1,p1)*; inverse relations carry the superscript "$^-$", as in $hasPart^-$. For further details the reader can refer to [2].

Example 4.1. From the OWL translation of the UML diagram in figure 6 we have that:
(1)$D\_subSystem1 \sqsubseteq DroneSystem1$ and
(2) $DroneSystem1 \sqsubseteq Device$
this implies: (3) $D\_subSystem1 \sqsubseteq Device$
In this example axioms (1) and (2) have been created via our UML-OWL mapping, whereas axiom (3) was inferred by the reasoner.

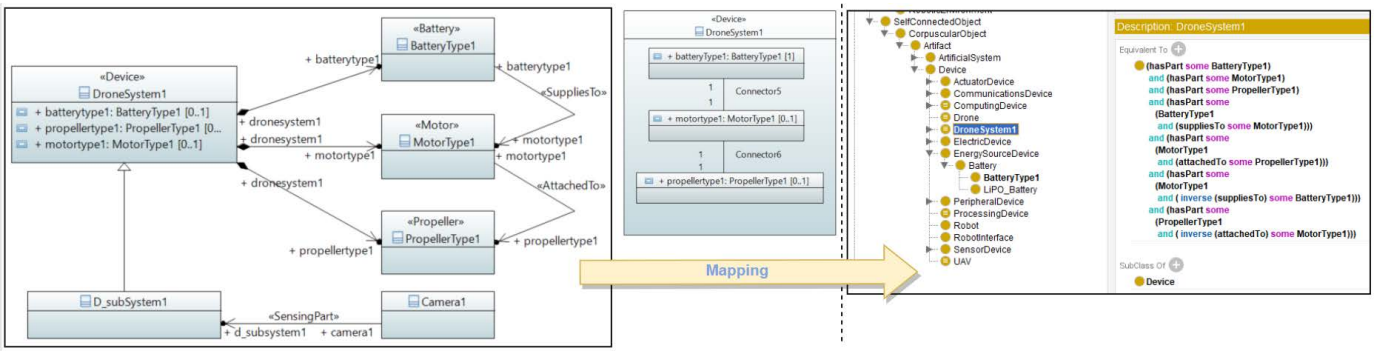

Figure 6: UML Class Diagram and UML Composite Structure Diagram, mapped into *ODrone*. An OWL Class and its Complex Class Definition are automatically generated by the approach.

Example 4.1 shows we can *infer* that *D_subSystem1* is a *Device*, even though we have never explicitly provided this information in the UML model. Likewise, note that on the diagram in figure 6, the class *Camera1* **has no stereotype**, but it *isConnectedTo D_subSystem1* by a *sensingPart* relation. As presented before, *ODrone* establishes that a *sensingPart* relation takes a *SensorDevice* as the source (domain) and a *Device* as a target (range), and thus using a reasoner we can infer that *Camera1* is indeed a *SensorDevice*, and therefore a *Device* (rendering implicit information explicit).

All the above defined terminology, classes, taxonomy and restrictions correspond to what is known as a T-Box in DL-ontologies (i.e. the terminology of the knowledge base) [3]. The T-Box talks about the types of objects that exist in a representation and the relationships between them. These can be seen as the rules that govern the (portion of) world we are modeling. Any violation of these rules would render the ontology inconsistent, and thus allowing only for valid representations to be accepted. Once these rules are expressed as axioms in the ontology, we can introduce assertions about the individuals (objects in the real world) that belong to those classes.

The classes and relations presented so far, serve as specifications of systems, but they do not represent instances of the systems themselves. Using the vocabulary of *ODrone* we create instances of the above defined *Devices*, and connect them. This can be done by specifying UML instances in Papyrus. The instance checking verification task here consists in evaluating whether or not the system being designed is an instance of *DroneSystem1*. To this end we create 4 individuals in example 4.2 along with their relations:

Example 4.2. The assertions (A-Box) of the instances that play the role of parts of a *DroneSystem1*:

$$Anonymous(d1) \quad BatteryType1(b_1) \quad MotorType1(m_1)$$
$$PropellerType1(p_1) \quad actuatingPart(m_1, d_1) \quad attachedTo(m_1, p_1)$$
$$energySourcePart(b_1, d_1) \quad suppliesTo(b_1, m_1) \quad actuatingPart(p_1, d_1)$$

The task here is to check whether $d1$ is an instance of $DroneSystem1$. The individuals in example 4.2 are underspecified w.r.t. $DroneSystem1$'s definition (right hand side of fig. 6), nevertheless when these individuals are added to the ontology, the reasoner infers *DroneSystem1(d1)*, as intended. This example, as simple as it appears, requires several inferences to assert $DroneSystem1(d_1)$. That is, *d1* has **all necessary and sufficient** parts of the required types, and these parts are interconnected in the intended ways. Note that in the definition of $DroneSystem1$ (fig. 6) we have never used the *part* relation directly, but because *actuatingPart* and *energySourcePart* are sub-properties of *part*, they also are *parts*. Moreover, the *part* relation, is a directed relation from the part to the composed system, whereas in our case we specified the relation from the point of view of the system having those parts (inverse direction). This example shows how inverse roles, class inheritance and relations inheritance are used to draw inferences about complex concept definitions, while preserving the semantics of the ontology.

Example 4.3. Our last example addresses the reuse of knowledge, and different reasoning methods available to provide meaningful inferences. CORA is a standard that has been developed by experts in the autonomous systems domain, where part of their knowledge has been formalized as logic axioms, but not all are formalized in the OWL version of the standard. In this example, we show how we formalize and reuse this knowledge to draw inferences about our UML model. First, CORA establishes that "A robot interacts with the world surrounding it through an interface.", and thus every *Robot* has a *RobotInterface*, formalized in CORA as:

$$inverse\ robotPart\ exactly\ 1\ RobotInterface$$

This definition formalizes the intended meaning of CORA *Robot*, but under the Open World Assumption (OWA), an individual connected to a *Device* through a *robotPart* relation, will be considered a *Robot* even if it does not have a *RobotInterface* defined. To detect whether or not the robot has an interface, we rely on SPARQL queries. Besides the inferences provided by an OWA reasoner, SPARQL will look for specific patterns in the underlying knowledge graph, and thus the query in figure 7 will capture only those *Robots* have a *RobotInterface* defined. In our example $d1$ is not consistent with the CORA definition of *Robot*, and thus we must add a *RobotInterface* to make it consistent. We accomplish this by stating:

$$RobotInterface(interface1),\ robotPart(interface1, d1)$$

Only then the model is compliant with CORA, which can be detected via SPARQL queries (fig. 7). This example shows how we can enforce that only compliant models are allowed.

Second, now that our UML model has a *robotPart* declared of type *RobotInterface*, there are additional inferences that can be drawn about our system, without adding any extra information. CORA

```
SELECT ?ind1 ?ind2
        WHERE {
                ?ind1 rdf:type ?aux1 .
                ?aux1 rdfs:subClassOf* cora:Robot .
                ?ind2 cora:robotPart ?ind1 .
                ?ind2 rdf:type ?aux2 .
                ?aux2 rdfs:subClassOf* cora:RobotInterface.
}
```

ind1	ind2
d1	interface1

Figure 7: SPARQL to filter those *Robots* that *do have* a *RobotInterface.*

establishes as well that *"The RobotInterface is a device composed by other devices that play the roles of sensing device, actuating device and communicating device"* and that, in turn, each one of these *robotParts* is as well a part of its *RobotInterface*. These conditions can be formalized as SWRL rules, as shown in figure 8. Because of

```
rparts:robotActuatingPart(?p, ?r) ^ cora-bare:RobotInterface(?i) ^ cora-bare:robotPart(?i, ?r)
-> sumo-cora:part(?p, ?i)
```

Figure 8: SWRL rules to enforce the *actuatingParts* of a *Robot*, are parts of its *RobotInterface*. The rules for *sensingParts* and *communicatingParts* are analogous.

these conditions, and having added that *d1* has an interface (step 1), we have that *m1* and *c1*, are both parts of *d1*'s interface. Thus providing additional information about our system by applying machine readable expert knowledge to our model.

5 CONCLUSIONS & FURTHER WORK

When tackling a specific domain (e.g. drone design), it is often not evident which terminology and semantics should be adhered to. This effectively increases the effort to implement a KB based system design. Vocabularies and standards are not only diverse, but not all are for free access, and only a few are available in the form of formal computer exploitable resources (e.g. OWL). We have provided an end-to-end solution for coupling DSL into standardized formal vocabularies, and their integration into MBSE tooling, effectively enabling the system designer to describe its system in terms of the ontology. Thanks to the bi-directional mappings, our approach also enables the export of the model as a W3C compliant representation. We have also demonstrated the feasibility of the approach through the implementation of a solution, and shown how the annotated model can be exploited through reasoning services. We have illustrated how expert knowledge can be further specified via SPARQL and SWRL to enforce constraints to UML designs.

As further work we aim to publish *ODrone* as well as the implementation and their documentation for free access. Given that the approach's concept is feasible, we are extending the reach of the use case and currently working on semantic interaction with other domains, such as safety, scenarios definition and simulation.

ACKNOWLEDGMENTS

This project has received funding from the ECSEL Joint Undertaking (JU) under grant agreement No 826276. The JU receives support from the European Union's Horizon 2020 research and innovation programme and France, Spain, Hungary, Italy, Germany.

REFERENCES

[1] Colin Atkinson and Kilian Kiko. 2005. A detailed comparison of UML and OWL. *None* (2005).
[2] Franz Baader. 2003. Appendix: description logic terminology. *The Description logic handbook: Theory, implementation, and applications* (2003), 485–495.
[3] Franz Baader, Diego Calvanese, Deborah McGuinness, Peter Patel-Schneider, Daniele Nardi, et al. 2003. *The description logic handbook: Theory, implementation and applications*. Cambridge university press.
[4] Mohammed Diab, Aliakbar Akbari, Muhayy Ud Din, and Jan Rosell. 2019. PMK—a knowledge processing framework for autonomous robotics perception and manipulation. *Sensors* 19, 5 (2019), 1166.
[5] Dov Dori and Hillary Sillitto. 2017. What is a system? An ontological framework. *Systems Engineering* 20, 3 (2017), 207–219.
[6] Mohamad Eid, Ramiro Liscano, and Abdulmotaleb El Saddik. 2007. A universal ontology for sensor networks data. In *2007 IEEE International Conference on Computational Intelligence for Measurement Systems and Applications*. IEEE, 59–62.
[7] Maged Elaasar and Nicolas Rouquette. 2022. http://www.opencaesar.io/oml/
[8] Hicham Elasri and Abderrahim Sekkaki. 2013. Semantic integration process of business components to support information system designers. *arXiv preprint arXiv:1302.1393* (2013).
[9] Colomb R. et al. 2006. The object management group ontology definition metamodel. In *Ontologies for software engineering and software technology*. Springer, 217–247.
[10] Manzoor S. et al. 2021. Ontology-based knowledge representation in robotic systems: A survey oriented toward applications. *Applied Sciences* 11, 10 (2021), 4324.
[11] Schlenoff C. et al. 2012. An IEEE standard ontology for robotics and automation. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 1337–1342.
[12] Vanherpen K. et al. 2016. Ontological reasoning for consistency in the design of cyber-physical systems. In *2016 1st International Workshop on Cyber-Physical Production Systems (CPPS)*. IEEE, 1–8.
[13] Ronald E Giachetti. 2015. Evaluation of the DoDAF meta-model's support of systems engineering. *Procedia Computer Science* 61 (2015), 254–260.
[14] Niklas Hallberg, Erland Jungert, and Sofie Pilemalm. 2014. Ontology for systems development. *International journal of software engineering and knowledge engineering* 24, 03 (2014), 329–345.
[15] Olszewska Joanna Isabelle et al. 2017. Ontology for autonomous robotics. In *2017 26th IEEE international symposium on robot and human interactive communication (RO-MAN)*. IEEE, 189–194.
[16] Eeva Järvenpää, Niko Siltala, Otto Hylli, and Minna Lanz. 2019. The development of an ontology for describing the capabilities of manufacturing resources. *Journal of Intelligent Manufacturing* 30, 2 (2019), 959–978.
[17] David Martín-Lammerding, Alberto Córdoba, José Javier Astrain, Pablo Medrano, and Jesús Villadangos. 2020. An ontology-based system to collect WSN-UAS data effectively. *IEEE Internet of Things Journal* 8, 5 (2020), 3636–3652.
[18] Marc H Meyer and Michael H Zack. 1996. The design and development of information products. *MIT Sloan Management Review* 37, 3 (1996), 43.
[19] Vitaliy Mezhuyev. 2014. Ontology based development of domain specific languages for systems engineering. In *2014 International Conference on Computer and Information Sciences (ICCOINS)*. IEEE, 1–6.
[20] Meriem Mejhed Mkhinini, Ouassila Labbani-Narsis, and Christophe Nicolle. 2020. Combining UML and ontology: An exploratory survey. *Computer Science Review* 35 (2020), 100223.
[21] Haradhan Mohajan. 2016. A comprehensive analysis of knowledge management cycles. (2016).
[22] Alberto Olivares-Alarcos et al. 2019. A review and comparison of ontology-based approaches to robot autonomy. *The Knowledge Engineering Review* 34 (2019).
[23] Fernando Silva Parreiras. 2011. Marrying model-driven engineering and ontology technologies: The TwoUse approach. (2011).
[24] E. Prestes, S. Fiorini, and J. Carbonera. 2014. Core Ontology for Robotics and Automation.
[25] Vitor Fortes Rey Sandro Rama Fiorini. 2015. IEEE1872-owl. https://github.com/srfiorini/IEEE1872-owl. OWL specification of the Core Ontology for Robotics and Automation (CORA) and other IEEE 1872-2015 ontologies..
[26] Moritz Tenorth and Michael Beetz. 2017. Representations for robot knowledge in the KnowRob framework. *Artificial Intelligence* 247 (2017), 151–169.
[27] LC Van Ruijven. 2013. Ontology for systems engineering. *Procedia Computer Science* 16 (2013), 383–392.
[28] Lan Yang et al. 2017. Towards a methodology for systems engineering ontology development—An ontology for system life cycle processes. In *2017 IEEE International Systems Engineering Symposium (ISSE)*. IEEE, 1–7.
[29] Lan Yang, Kathryn Cormican, and Ming Yu. 2019. Ontology-based systems engineering: A state-of-the-art review. *Computers in Industry* 111 (2019), 148–171.