A Methodology for Knowledge Integration and Acquisition in Model-Based Systems Engineering

Luis Palacios Medinacelli[a], Florian Noyrit[b] and Chokri Mraidha[c]
Université Paris-Saclay, CEA, List. Palaiseau, France
{*luis.palacios, florian.noyrit, chokri.mraidha*}*@cea.fr*

Keywords: Domain-Specific Ontologies, Knowledge Integration, MBSE, Semantic Interoperability.

Abstract: In Model-Based System Engineering (MBSE) systems are represented as models using a predefined meta-language such as SysML that hides some of the complexity behind the specification of a system, and provides experts with a rich syntax to define, share and constrain these models. Even though current MBSE design tools are sophisticated and support expressive meta-languages, these tools have limited capabilities on the detection of semantic errors, the integration of expert knowledge, or the ability to formalize the knowledge from the expert's design. Our work addresses these limitations by annotating the SysML models with domain-specific ontologies. Enabling this interaction not only makes the ontology's semantics available to the tooling environment, but the UML model specified by the designer can be translated into Ontology Web Language format (OWL), generating a system specification in terms of the ontology. These "annotated models" are suitable for reasoning tasks, like consistency check and instance checking. We particularly ensure the annotated model is a consistent extension of the domain-specific ontology, thus formalizing the expert's knowledge as a sub-ontology. This extended ontology can be re-used and shared to evaluate and constrain further models, or be itself evaluated by semantic-compatible tools. In this article we present our approach for an OWL-MBSE integration, and show its feasibility via an implementation in the UAVs domain.

1 INTRODUCTION

Complex systems design, like Autonomous or Cyber-Physical Systems, require specific design and engineering techniques to ensure the resulting systems comply with their 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 SysML[1], a UML[2] profile for system engineering. These languages have a very rich expressiveness. The sophistication of current modeling tools is correlated to this high expressiveness, thus requiring high expertise in the tool's specific representation language. Moreover, new modeling 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 formal syntax of MBSE, its semantics are expressed in natural language, which poses a limitation for interpretation, sharing and reuse of these models. Moreover, because different stakeholders and different projects may define the same systems differently, semantic interoperability and the construction of knowledge bases from these system models remains challenging. The use of formal semantics is essential for explicit, shareable, and reusable knowledge representation (Yang et al., 2019).

There exists extensive research (Yang et al., 2019; Atkinson and Kiko, 2005; Berardi et al., 2005; Parreiras, 2011) on the benefits and potential of the interaction and integration of MBSE and ontologies. These vary from providing reference concepts and the definition of homogeneous terminologies, to allowing for machine-readable axioms and integration, exchange and reuse of knowledge, among others. It is also noted in these works the challenges for a successful integration, among which we find: the (un)availability of the required ontologies along with

[a] https://orcid.org/0000-0002-9413-4119
[b] https://orcid.org/0000-0001-5947-7258
[c] https://orcid.org/0000-0003-2993-5734
[1] https://www.sysml.org/
[2] https://www.uml.org/

Medinacelli, L., Noyrit, F. and Mraidha, C.
A Methodology for Knowledge Integration and Acquisition in Model-Based Systems Engineering.
DOI: 10.5220/0012233900003598
In *Proceedings of the 15th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management (IC3K 2023) - Volume 2: KEOD*, pages 264-271
ISBN: 978-989-758-671-2; ISSN: 2184-3228

their documentation; the different levels of abstraction and viewpoints, where the domain and scope of the ontology needs considerable effort to be aligned to a new application domain; the different interpretations of a given language construct depending on the viewpoint; and the gap between theoretically focused approaches and real-world applications.

To enable the interaction between different viewpoints and representations, and to enable the application and reuse of ontological knowledge within MBSE, in this article we present a formal approach to integrate and extract knowledge, expressed as an ontology, into/from MBSE tools.

Our approach emphasizes in the following challenges : 1) The integration of standardized and domain-specific languages, 2) The definition of the components a bidirectional mapping (OWL-UML) should consider, guided by the expressivity of specific description logic languages, 3) The ability to capture complex system structures and describe them as complex ontological definitions 4) the reuse of the captured knowledge.

In this paper we present the formalization of our approach, to integrate standardized domain-specific ontologies into MBSE tooling environments, and show its feasibility via an implementation in the UAVs (Unmanned Aerial Vehicles) domain.

The rest of this paper is organized as follows: in section 2 we present the works related to ontologies and MBSE. In section 3 we present our approach. In section 4 we present an implementation of the approach, motivated by the UAVs use case. Finally, we present our conclusions in section 5.

2 RELATED WORKS

In this section, we present some of the most relevant works regarding ontologies, UML/SysML and MBSE technologies. These works present the motivations, state of the art, challenges, and envisaged benefits from the interaction of the aforementioned technologies. 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 (Parreiras, 2011; Elasri and Sekkaki, 2013). There is also the risk of loss of information when exchanging between heterogeneous 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.

The work in (Atkinson and Kiko, 2005) evidences that there is a lack of a complete mapping between the constructs of the two languages. Although, within the terminology of an ontology, there might be specific concepts and relations that suit specific UML constructs, allowing for a more complete mapping.

The work in (Berardi et al., 2005) explores the expressivity and reasoning complexity in UML diagrams, and the work in (Parreiras, 2011) aims to provide an integrated approach for UML class-based modeling and ontology modeling. There are several areas of system engineering (SE) knowledge where research between ontologies and SE has been conducted (Yang et al., 2019): *System Fundamentals*, *System engineering Standards*, *Generic Life Cycle Stages*, *Representing Systems with Models*, *Engineered System Contexts* and *System Engineering Management*. The works considered in (Yang et al., 2019) 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.

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 (Vanherpen, 2016). 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 (Mezhuyev, 2014). And, by using a predefined ontology, it is possible to reduce the number of misinterpretations within projects (Hallberg et al., 2014).

3 APPROACH

Let us first introduce some preliminary notions required to define our approach. In the following we assume the reader is familiar with OWL[3] ontologies and Description Logics DLs. DL is a family of FOL languages. Thanks to a carefully bounded expressivity, some of them can provide tractable reasoning ser-

[3]https://www.w3.org/TR/owl2-overview/

vices, such as consistency or instance checking.

In our work we consider the description logics $\mathcal{ALCHI}^{\mathcal{D}}$. The rationale behind this choice lies in that we aimed to at least provide $\mathcal{ALC}'s$ expressivity, plus enabling the extension of an ontology's taxonomy, and the use of datatypes. Next, we need to recall the corresponding syntax definitions of $\mathcal{ALCHI}^{\mathcal{D}}$. For further details the reader may refer to (Baader, 2003) and (Baader et al., 2017).

Let N_C, N_R, N_D, and N_I be pairwise disjoint (nonempty) sets of *concept names*, *object property names*, *data property names*, and *individual names*, respectively. We denote by N_R^- the set of inverses of all $r \in N_R$. A *role* is an element of $N_R \cup N_R^- \cup N_D$. *Concepts* are defined as follows: every $\phi \in N_C$ is a concept, if $o_1, o_2, \dots o_n \in N_I$ then $\{o_1, o_2, \dots o_n\}$ is a concept. If ϕ, ϕ_1 and ϕ_2 are concepts, then $(\phi_1 \sqcup \phi_2)$,$(\phi_1 \sqcap \phi_2)$, and $(\neg\phi)$ are concepts (called conjunction, disjunction, and negation, respectively). If $r \in N_R \cup N_R^-$ then $\exists r.\phi, \forall r.\phi$ are concepts. If D is a datatype and $s \in N_D$, then $\exists s.D, \forall s.D$ are concepts as well. We write $\top$ and $\bot$ to abbreviate the concepts $(\phi \sqcup \neg\phi)$ and $(\phi \sqcap \neg\phi)$, respectively. We eliminate parentheses as usual.

Axioms. An axiom in $\mathcal{ALCHI}$ has one of the following forms: 1) $\phi \sqsubseteq \psi$ (called concept inclusion axiom), where ϕ and ψ are concepts; (2) $r \sqsubseteq s$ (called role inclusion axiom), where either $r, s \in N_R \cup N_R^-$ or $r, s \in N_D$ (3) $\phi(a)$ (called concept membership axiom), where ϕ is a concept and $a \in N_I$; (4) $r(a,b)$ (resp., $d(a,b)$) (called role membership axiom), where $r \in N_R \cup N_R^-$ (resp., $d \in N_D$) and $a, v \in N_I$ (resp., $a \in N_I$ and v is a data value). A *Knowledge Base L* is a finite set of axioms.

Terminological Axioms. A general concept inclusion (GCI) has the form $C \sqsubseteq D$ where C and D are concepts. We write $C \equiv D$ when $C \sqsubseteq D$ and $D \sqsubseteq C$. A *T-Box* is a finite set of GCIs.

Assertional Axioms. A concept assertion is a statement of the form $C(a)$ where $a \in N_I$ and C is a concept. A role assertion is a statement of the form $r(a,b)$ where $a, b \in N_I$ and r is a role. An *A-Box* is a finite set of assertional axioms.

A *DL knowledge base* (KB) is a pair $(\mathcal{T}, \mathcal{A})$ for T-Box $\mathcal{T}$ and A-Box $\mathcal{A}$. Since knowledge bases encode ontologies, in the rest of this paper we refer to a DL knowledge base and an ontology indistinctly.

To denote our concepts, roles and individuals, we adhere to the notation in (Baader et al., 2003), where predicates starting in uppercase, like *Device* or *SensorDevice*, denote concepts; predicates starting in lower case, like *sensingPart*, denote roles. Individuals are denoted by terms in lowercase, and possibly carrying a subscript number. Individuals are asserted as instances of a class, as parameters of unary predicates, like $Device(d_1)$, or as taking part in a role, like $hasPart(d_1, p_1)$; inverse roles carry the superscript "$^-$", as in $hasPart^-$.

3.1 Approach Definitions

In the following let O_{up}, O_d be two ontologies, where $O^C = \{C_1, C_2..., C_n\}$ is the set of concepts names in ontology O, and $O^r = \{r_1, r_2..., r_m\}$ is the set of object and data property names in O.

Definition 1 (Integration Specification). *Given two ontologies O_{up}, O_d, an* Integration Specification *of O_d into O_{up} is a pair, of sets of pairs, of the form:*

$$\begin{array}{ll} O_d \to O_{up} & = \langle IC, IR\rangle \text{ where:} \\ IC & = \{\langle C_1^d, C_1^{up}\rangle, \langle C_2^d, C_2^{up}\rangle, \dots, \langle C_n^d, C_n^{up}\rangle\} \\ IR & = \{\langle r_1^d, r_1^{up}\rangle, \langle r_2^d, r_2^{up}\rangle, \dots, \langle r_m^d, r_m^{up}\rangle\} \end{array}$$

, for $m, n \geq 0$, and each $C_i^d \in O_d^C$, $r_i^d \in O_d^r$, $C_i^{up} \in O_{up}^C$, and $r_i^{up} \in O_{up}^r$.

Example 1 (Upper and Domain-Specific Ontologies). *Let O_{CORA} and O_{Drone} be two ontologies, with*

$$\begin{array}{ll} O_{CORA}^C & = \{Device, ActuatorDevice\} \\ O_{CORA}^r & = \{part, robotPart, interactsWith\} \\ O_{Drone}^C & = \{SensorDevice, Camera, Motor, \\ & Propeller\} \\ O_{Drone}^r & = \{hasPart, isConnectedTo\} \end{array}$$

An Integration Specification *of O_{Drone} into O_{CORA} can be defined as:*

$O_{Drone} \to O_{CORA} =$
$\langle\{\langle SensorDevice, Device\rangle, \langle Motor, ActuatorDevice\rangle\},$
$\{\langle hasPart, part^-\rangle, \langle isConnectedTo, interactsWith\rangle\}\rangle$

Definition 2 (Union Ontology). *Given two ontologies O_d, O_{up} and an* integration specification *$O_d \to O_{up}$, we define the* union ontology *as:*

$$U_{O_d, O_{up}} = O_d \cup O_{up} \cup \{\bigcup_{i=1}^{n} C_i^d \sqsubseteq C_i^{up}\} \cup \{\bigcup_{j=1}^{m} r_j^d \sqsubseteq r_j^{up}\}$$

Where each $\langle C_i^d, C_i^{up}\rangle \in IC$ with $n = |IC|$ and, each $\langle r_j^d, r_j^{up}\rangle \in IR$ with $m = |IR|$.

Example 2 (Union Ontology). *Given the* integration specification *$O_{Drone} \to O_{CORA}$ defined in example 1, the* union ontology *of O_{CORA} and O_{Drone} is:*

$$\begin{array}{rl} U_{O_{Drone}, O_{CORA}} & = O_{Drone} \cup O_{CORA} \cup \\ \{SensorDevice & \sqsubseteq Device, \\ Motor & \sqsubseteq ActuatorDevice\} \cup \\ \{hasPart & \sqsubseteq part^-, \\ isConnectedTo & \sqsubseteq interactsWith\} \end{array}$$

Definition 3 (Consistent Integration)**.** *Given two ontologies O_d, O_{up} , an integration specification $O_d \rightarrow O_{up}$ and the union ontology $U_{O_d,O_{up}}$, we say that $O_d \rightarrow O_{up}$ is consistent if:*

$$U_{O_d,O_{up}} \models \top$$

i.e. $U_{O_d,O_{up}}$ is satisfiable.

The intended meaning of the concepts and relations from the ontology, have to be preserved throughout the transformation into UML and back to OWL. Because of the variety of the constructs in both formalisms, their correlation is not only not evident, but case-dependent. Moreover, there are specific sets of constructs and diagrams in UML for describing specific systems and entities. The UML specification[4] defines the UML meta-model. The root concepts : *Element* and *Relationship*, provide the basis for all other modeling concepts in UML. In the rest of this article, we prefix elements of the UML language with "*uml::*" (like *uml::Class* or *uml::Component*) and properties of those elements are separated by "*::*" as in *uml::Element::ownedElement*. For our work, the elements in the UML language, are considered as UML constructs, that can be specialized and/or instantiated by the system's designer to specify his model. We target these constructs to define a mapping between OWL and UML.

Definition 4 (A mapping function $\mu^{O \rightarrow UML}$)**.** *Let*

$$\sigma_{UML} = \{\mathcal{U}_1, \mathcal{U}_2, \dots \mathcal{U}_n\}$$

be the set of names of all constructs in the UML meta-model (i.e. UML language), and let O be an ontology, with the set of concept names O^C, and the set of role names O^r.

We define the mapping:

$$\mu^{O \rightarrow UML} = \{\mu_C, \mu_{OP}, \mu_{DP}\}$$

with:

$$\begin{aligned} \mu_C &= \{\langle C_1, \mathcal{U}_1\rangle, \langle C_2, \mathcal{U}_2\rangle, \dots, \langle C_m, \mathcal{U}_m\rangle\} \\ \mu_{OP} &= \{\langle r_1, \mathcal{U}_1\rangle, \langle r_2, \mathcal{U}_2\rangle, \dots, \langle r_n, \mathcal{U}_n\rangle\} \\ \mu_{DP} &= \{\langle r_1, \mathcal{U}_1\rangle, \langle r_2, \mathcal{U}_2\rangle, \dots, \langle r_p, \mathcal{U}_p\rangle\} \end{aligned}$$

where :

1)$\forall \langle x,y\rangle \in \mu_C \mid x \in O^C, y \in \sigma_{UML}, \langle z,y\rangle \notin \{\mu_{OP}, \mu_{DP}\}$
2)$\forall \langle x,y\rangle \in \mu_{OP} \mid x \in O^r, y \in \sigma_{UML}, \langle z,y\rangle \notin \{\mu_C, \mu_{DP}\}$
3)$\forall \langle x,y\rangle \in \mu_{DP} \mid x \in O^r, y \in \sigma_{UML}, \langle z,y\rangle \notin \{\mu_C, \mu_{OP}\}$

Example 3 (Mapping $\mu^{O \rightarrow UML}$)**.** *where:*

$\mu_C = \{\langle Device, uml :: Class\rangle,$
$\langle Device, uml :: Component\rangle\}$
$\mu_{OP} =$
$\{\langle hasPart, uml :: Element :: ownedElement\rangle,$
$\langle isConnectedTo, uml :: Association\rangle\}$
$\mu_{DP} = \{\}$

We can also see that conditions 1), 2) and 3) from definition 4 are satisfied.

Definition 4 allows single ontology concepts (and relations), to be mapped to multiple UML constructs, and vice-versa. Note that the mapping $\mu^{O \rightarrow UML}$ does not map names in O to actual UML models elements, but to its meta-language. The mapping $\mu^{O \rightarrow UML}$ allows to select a relevant subset of names in O, and defines the elements in a UML model that *can be typed* by these names. This is achieved by creating UML *stereotypes* for each entry in the mapping. Stereotypes are a UML mechanism to extend its language, that allows to specialize an element, and are defined in a UML profile. Each stereotype can be applied to a restricted set of elements. Thus, the mapping is not *applied* to the UML model automatically. It is the system designer who, in the end, defines which ontology concepts correspond to (or generalise) elements in his model (i.e. by applying the stereotypes). The annotated UML model might further specialize the classes and relations from the ontology. It might represent single concepts as composite structures or as properties of classes. Thus an "inverse" mapping, from an annotated UML model to O, will indeed depend on $\mu^{O \rightarrow UML}$, but we can not simply "inverse it" to get a meaningful representation of the UML model. To consider the multiple representations that a UML model can have, the mapping from UML to OWL needs to distinguish instances from concepts. It also needs to take into account the specialization mechanisms in UML, and it is desirable that complex structures and complex concept definitions are handled.

Definition 5 (A mapping function $\mu^{UML \rightarrow O}$)**.** *Let $\mathcal{M}$ be a UML model, and let*

$$\sigma_{\mathcal{M}} = \{\mathcal{M}_1, \mathcal{M}_2, \dots \mathcal{M}_n\}$$

be the set of names of all elements in $\mathcal{M}$. Then the mapping from UML model $\mathcal{M}$ to ontology O is defined by:

$$\mu^{UML \rightarrow O} = \{\mu'_{\sqsubseteq}, \mu'_{\equiv}, \mu'_{OP}, \mu'_{DP}, \mu'_{C(i)}, \mu'_{r(a,b)}\}$$

with:

$$\begin{aligned} \mu'_{\sqsubseteq} &= \{\langle \mathcal{M}_1, C_1\rangle, \langle \mathcal{M}_2, C_2\rangle, \dots, \langle \mathcal{M}_i, C_i\rangle\} \\ \mu'_{\equiv} &= \{\langle \mathcal{M}_1, C_1\rangle, \langle \mathcal{M}_2, C_2\rangle, \dots, \langle \mathcal{M}_j, C_j\rangle\} \\ \mu'_{OP} &= \{\langle \mathcal{M}_1, r_1\rangle, \langle \mathcal{M}_2, r_2\rangle, \dots, \langle \mathcal{M}_k, r_k\rangle\} \\ \mu'_{DP} &= \{\langle \mathcal{M}_1, r_1\rangle, \langle \mathcal{M}_2, r_2\rangle, \dots, \langle \mathcal{M}_l, r_l\rangle\} \\ \mu'_{C(i)} &= \{\langle \mathcal{M}_1, C(i)_1\rangle, \langle \mathcal{M}_2, C(i)_2\rangle, \dots, \\ &\quad \langle \mathcal{M}_m, C(i)_m\rangle\} \\ \mu'_{r(a,b)} &= \{\langle \mathcal{M}_1, r(a,b)_1\rangle, \langle \mathcal{M}_2, r(a,b)_2\rangle, \\ &\quad \dots, \langle \mathcal{M}_n, r(a,b)_n\rangle\} \end{aligned}$$

with $i,j,k,l,m,n \geq 0$. Where each C_i is a (possibly complex) concept in $\mathcal{ALCHI}^{\mathcal{D}}$, each r_i is a role *in*

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

O, and where $\langle \mathcal{M}_i, x\rangle$ *and* $\langle \mathcal{M}_i, y\rangle$ *can not occur, if* $x \neq y$.

Definition 6 (The application of a mapping function $\mu^{UML\to O}$). *Given an ontology O, a mapping* $\mu^{UML\to O}$ *and an UML model M, the* application of $\mu^{UML\to O}$ *to M yields the set of axioms* O_{UML}*, written as :*

$$O_{UML} = M^{\mu^{UML\to O}}$$

, recursively defined by:

$$\begin{array}{lcl}
\forall\langle A,B\rangle \in \mu_{\sqsubseteq} & \Leftrightarrow & \{A \sqsubseteq B\} \in O_{UML}\\
\forall\langle A,B\rangle \in \mu_{\equiv} & \Leftrightarrow & \{A \equiv B\} \in O_{UML}\\
\forall\langle A,B\rangle \in \mu_{OP} & \Leftrightarrow & \{A \sqsubseteq B\} \in O_{UML}\\
\forall\langle A,B\rangle \in \mu_{DP} & \Leftrightarrow & \{A \sqsubseteq B\} \in O_{UML}\\
\forall\langle A,C(i)\rangle \in \mu_{C(i)} & \Leftrightarrow & \{C(i)\} \in O_{UML}\\
\forall\langle A,r(a,b)\rangle \in \mu_{r(a,b)} & \Leftrightarrow & \{r(a,b)\} \in O_{UML}
\end{array}$$

Example 4 (Definition and Application of $\mu^{UML\to O}$). *Consider a UML model M′, a simplified version of the model in figure 1, containing the following elements:*

$$\begin{array}{ll}
\sigma_{M'} = & \{DroneSystem_1, BatteryType_1, MotorType_1,\\
& PropellerType_1, D_subSystem1, Camera1,\\
& attachedTo, is_1, s_1.ownedElement.m_1,\\
& m_1, p_1, s_1, s_1.ownedElement.p_1\}
\end{array}$$

, we define the mapping $\mu^{UML\to O}$ *as:*

$$\begin{array}{ll}
\mu_{\sqsubseteq} & = \{\langle PropellerType1, Propeller\rangle,\\
& \langle MotorType1, Motor\rangle,\\
& \langle DroneSystem1, Device\rangle\}\\
\mu_{\equiv} & = \{\langle DroneSystem_1, hasPart.PropellerType1\\
& \sqcap hasPart.MotorType1 \sqcap\\
& hasPart.BatteryType1\rangle\}\\
\mu_{OP} & = \{\langle attachedTo, connectedTo\rangle\}\\
\mu_{DP} & = \{\}\\
\mu_{C(i)} & = \{\langle m_1, MotorType1(m_1)\rangle,\\
& \langle p_1, PropellerType1(p_1)\rangle,\\
& \langle s_1, DroneSystem1(s_1)\rangle\}\\
\mu_{r(a,b)} & = \{\langle is_1, attachedTo(m_1,p_1)\rangle,\\
& \langle s_1 :: ownedElement :: m_1, hasPart(s_1,m_1)\rangle,\\
& \langle s_1 :: ownedElement :: p_1, hasPart(s_1,p_1)\rangle\}
\end{array}$$

Then, the application *of* $\mu^{UML\to O}$ *to model M yields:*

$$\begin{array}{ll}
O_{UML} = \{ & \\
PropellerType1 & \sqsubseteq Propeller,\\
MotorType1 & \sqsubseteq Motor,\\
DroneSystem1 & \sqsubseteq Device,\\
DroneSystem1 & \equiv hasPart.PropellerType1 \sqcap\\
& hasPart.MotorType1 \sqcap\\
& hasPart.BatteryType1,\\
attachedTo & \sqsubseteq connectedTo,\\
MotorType1(m_1), & PropellerType1(p_1),\\
DroneSystem1(s_1), & attachedTo(m_1,p_1),\\
hasPart(s_1,m_1), & hasPart(s_1,p_1)\}
\end{array}$$

Definition 7 (A System Specification T-Box). *Given an UML model M and the mapping* $\mu^{UML\to O}$*, the System Specification T-Box* $\mathcal{T}_{UML}$ *is the result of applying the restricted mapping:*

$$\mu^{\mathcal{T}} = \{\mu_{\sqsubseteq}, \mu_{\equiv}, \mu_{OP}, \mu_{DP}\}$$

to M.

$$\mathcal{T}_{UML} = M^{\mu^{\mathcal{T}}}$$

Example 5 (System Specification T-Box). *Given the UML model M′ and the mapping* $\mu^{\mathcal{T}}$*, the* application *of* $\mu^{\mathcal{T}}$ *to M yields:*

$$\begin{array}{rl}
\mathcal{T}_{UML} = \{ & \\
PropellerType1 & \sqsubseteq Propeller,\\
MotorType1 & \sqsubseteq Motor,\\
DroneSystem1 & \sqsubseteq Device,\\
DroneSystem1 & \equiv hasPart.PropellerType1 \sqcap\\
& hasPart.MotorType1 \sqcap\\
& hasPart.PropellerType1,\\
attachedTo & \sqsubseteq connectedTo\}
\end{array}$$

Definition 8 (A UML System Instance). *Given an UML model M and the mapping* $\mu^{UML\to O}$*, a* system instance A-Box $\mathcal{A}_{UML}$*, is the result of applying the restricted mapping:*

$$\mu^{\mathcal{A}} = \{\mu_{C(i)}, \mu_{r(a,b)}\}$$

to M.

$$\mathcal{A}_{UML} = M^{\mu^{\mathcal{A}}}$$

Example 6 (System Instance). *Given the UML model M and the mapping* $\mu^{\mathcal{A}}$*, the* application *of* $\mu^{\mathcal{A}}$ *to M yields:*

$$\begin{array}{l}
\mathcal{A}_{UML} = \{MotorType1(m_1), PropellerType1(p_1),\\
DroneSystem1(s_1), attachedTo(m_1,p_1),\\
hasPart(s_1,m_1), hasPart(s_1,p_1)\}
\end{array}$$

Definition 9 (Consistent System Specification). *Given an ontology O, an UML model M and a mapping* $\mu^{UML\to O}$*, a* consistent system specification *is a T-Box* T_{UML} *s.t.*

$$O \cup \mathcal{T}_{UML} \models \top$$

Definition 10 (System Specification Model). *Given an ontology O, an UML model M, a mapping* $\mu^{UML\to O}$*, and a* consistent system specification T_{UML}*,a* system specification model *for the consistent specification* $\mathcal{T}_{UML}$ *is an A-Box* $\mathcal{A}_{UML}$ *s.t.*

$$O \cup \mathcal{T}_{UML} \cup \mathcal{A}_{UML} \models \top$$

4 IMPLEMENTATION

The process of integration of ontologies and an UML system model is illustrated in Figure 2. In the implementation we target Papyrus[5] (an open-source MBSE

[5] https://www.eclipse.org/papyrus/index.php

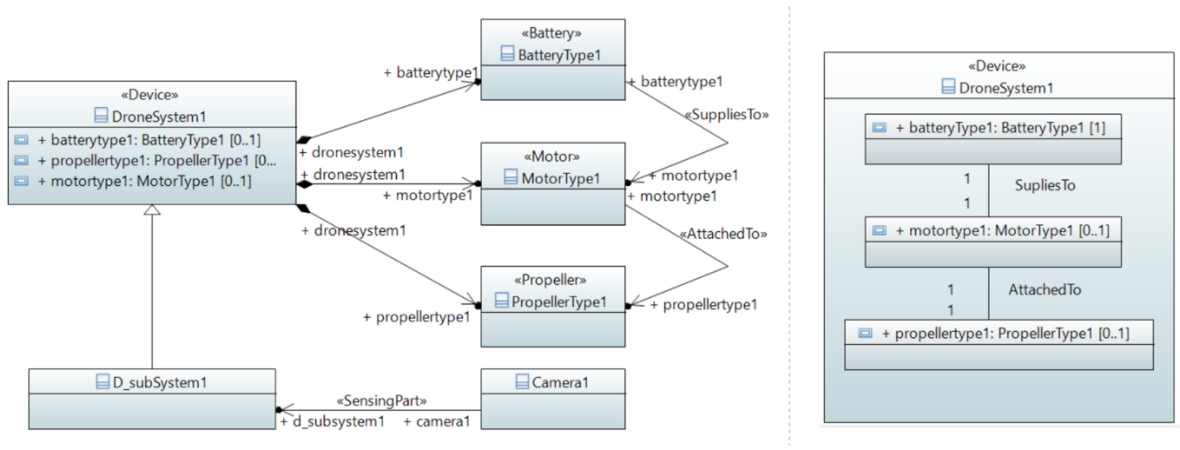

Figure 1: On the left, the UML Class Diagram for model $\mathcal{M}$ in example 4, on the right the Composite Structure Diagram for *DroneSystem*1.

tool) and use UML Profiles to make the terminology of the ontology available to the system's designer. The workflow is divided into four tasks, represented by the circles in the diagram. The inputs and outputs of these processes are depicted as color-coded files with extension .oml (blue) or .uml (yellow). In the following, we first explain the workflow and next, we detail the construction of the mappings.

In Figure 2 on the top left we have two ontologies: CORA.owl = O_1 and ODrone.owl = O_2. ODrone.owl is the Domain-Specific Ontology (Medinacelli et al., 2022). This is the specific vocabulary used in the target domain of interest. Our implementation is encompassed within the autonomous systems domain, and specifically the UAVs subdomain. The relevant definitions in the domain-specific ontology need to be mapped to UML. Specific ODrone concepts can be asserted as specializations of UML classes, and ODrone object and data properties can be asserted as specializations of UML associations.

Task 1 outputs a consistent integration of O_2 into O_1, namely $Union_{O_2,O_1}$ (see section 3). The union ontology $Union_{O_2,O_1}$ consistently integrates the domain-specific ontology ODrone into the upper and standardized ontology CORA. Thus providing a standardized domain-specific ontology $Union_{O_2,O_1}$. Task 2 maps concepts and roles from $Union_{O_2,O_1}$ to a UML profile thanks to the mapping specification $\mu^{O \to UML}$, making the ontology semantics available to the system designer in the MBSE tooling.

Indeed, the profile can now be used to model a system in task 3. Thus it is the system designer who *applies* the mapping $\mu^{O \to UML}$ to a specific UML model M. Once the annotated model is ready, the *application* of the "inverse" mapping $\mu^{UML \to O}$ takes place in task 4. This process yields O_{UML}, an *ODrone*/OWL compliant representation of the UML system, from where

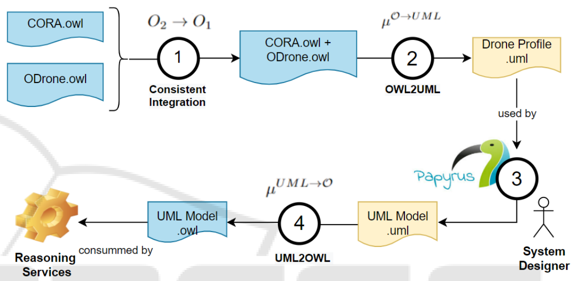

Figure 2: The workflow and resources for the implementation of the approach.

a *system specification* T_{UML} and a *system instance* A_{UML} can be obtained. The translated model O_{UML} 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, in the same manner as in (Medinacelli et al., 2022).

4.1 Mappings

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 (Mkhinini et al., 2020), it is common to target Class Diagrams constructs due to its evident similarity with ontology constructs (e.g. classes, relations). Furthermore, Composite Structure diagrams allow further specifying the relations between the parts of a system. Using constructs in these two diagrams, the designer can describe the entities that play a role in his design, as well as specific composite structures that carry interconnected parts.

In the ODrone implementation, we aim to describe the physical components of UAVs design, that is, the parts of the system and how these are interconnected. UML as well as ODrone, consider a "composition relation" named *part*. The ODrone:hasPart object property's meaning is analogous to the uml::AggregationKind::composite property. Both describe composition of complex structures by their *parts*. This correspondence between ODrone and UML, is taken into account by algorithm 1. It shows how to implement a mapping that considers case-specific correspondences (between the ontology and UML) and allows for the description of complex composed UML structures using the ontology concepts and relations. The output of algorithm 1, is the restricted mapping $\mu^{\mathcal{T}}$ with which we can obtain a system specification $\mathcal{T}_{UML}$. Whereas algorithm 2, constructs the mapping $\mu^{\mathcal{A}}$, which can be applied to a UML model M to obtain a system instance $\mathcal{A}_{UML}$. These two algorithms are presented next.

Recall:

$$\mu^{\mathcal{T}} = \{\mu_{\sqsubseteq}, \mu_{\equiv}, \mu_{OP}, \mu_{DP}\}$$

In algorithm 1 we first (1) initialize the sets $\mu_{\sqsubseteq}, \mu_{\equiv}, \mu_{OP}, \mu_{DP}$ to be empty. Then for each uml::Element in the model M (2) we verify if it is a (3) uml::Class. Regardless of whether the uml::Class has a stereotype (4), we add the pair $\langle uml :: Class, owl : Thing\rangle$ to $\mu_{\sqsubseteq}$, thus every uml::Class is mapped as subclass of owl:Thing. Line (5) handles the case of specialized classes, and in (9) we state that each stereotyped class is subsumed by its stereotype. In line (11) we introduce the set $CCD_E = \{\}$ (Complex Concept Description for E), this set will be incrementally constructed to hold all stereotyped relations from the class (12-16), and all its uml::CompositeAggregation (19-23) in the form of a complex concept definitions. Note in line (22) that each uml::CompositeAggregation is mapped to odrone:hasPart. Thus relating a uml::Structure with a domain-specific relation, existing only in ODrone. Line (26) states that the pair $\langle E, CCD_E\rangle$ belongs to $\mu_{\equiv}$, effectively providing a complex concept definition for E, in terms of ODrone. Finally, in (29) a subsumption relation is added to μ_{OP} for each stereotyped uml::Relationship. Note that this specific mapping allows only for one stereotype per relation.

Let us now introduce the algorithm 2 to construct the mapping $\mu^{\mathcal{A}}$. Recall:

$$\mu^{\mathcal{A}} = \{\mu_{C(i)}, \mu_{r(a,b)}\}$$

Algorithm 2 constructs the sets composing $\mu^{\mathcal{A}}$, and aggregates them into the output $\mu^{\mathcal{A}}$. Algorithm 2 is more straightforward, since we target only two UML elements, both specializations of the same top element uml::InstanceSpecification.

Input: M, $\mu^{O \to UML}$
Output: $\mu^{\mathcal{T}}$
1: $\mu_{\sqsubseteq} = \mu_{\equiv} = \mu_{OP} = \mu_{DP} = \{\}$
2: **for** each uml::Element E $\in$ M **do**
3:   **if** E isA uml::Class **then**
4:     $\langle E, owl : Thing\rangle \in \mu_{\sqsubseteq}$
5:     **if** E specializaionOf B $\in$ M **then**
6:       $\langle E, B\rangle \in \mu_{\sqsubseteq}$
7:     **end if**
8:     **for** each E.AppliedStereotypes S $\in$ M **do**
9:       $\langle E, S\rangle \in \mu_{\sqsubseteq}$
10:     **end for**
11:     $CCD_E = \{\}$
12:     **for** each E.getRelationships R $\in$ M **do**
13:       **if** R isA uml::Association **and** R.AppliedStereotypes $\neq \emptyset$ **then**
14:         b= R.target
15:         S= R.AppliedStereotype
16:         $CCD_E = CCD_E \sqcap \{\exists S.b\}$
17:       **end if**
18:     **end for**
19:     **for** each E.getOwnedElements OE $\in$ M **do**
20:       **if** OE isA uml::CompositeAggregation **then**
21:         b= OE.getType
22:         S= odrone:hasPart
23:         $CCD_E = CCD_E \sqcap \{\exists S.b\}$
24:       **end if**
25:     **end for**
26:     $\langle E, CCD_E\rangle \in \mu_{\equiv}$
27:   **end if**
28: **end for**
29: **if** E isA uml::Relationship **and** |E.AppliedStereotypes|= 1 **then**
30:   S= R.AppliedStereotype
31:   $\langle E, S\rangle \in \mu_{OP}$
32: **end if**
33: **return** $\mu^{\mathcal{T}} = \{\mu_{\sqsubseteq}, \mu_{\equiv}, \mu_{OP}, \{\}\}$

Algorithm 1: Construction of $\mu^{\mathcal{T}}$.

In algorithm 2 we first (1) initialize the sets $\mu_{C(i)}, \mu_{r(a,b)}$ to be empty. Then for each uml::Element in the model M (2) we verify if it is a (3) uml::InstanceSpecification. The way UML assigns a class to an instance is through its classifiers. Lines (4-7) state that for every classifier C of an instance E, the pair $\langle E, C(E)\rangle$ belongs to the set $\mu_{C(i)}$. Note that this process allows for multiple classifiers for the same instance, which is intended in OWL. In the case where an instance has no classifier (for example an anonymous individual which just participates in a relation) line (9) assigns to instance E the most general concept in CORA, i.e. $cora : Entity$. If it is the case the element E is an instance specification of a relationship

Input: M, $\mu^{O \rightarrow UML}$
Output: $\mu^{\mathcal{A}}$
1: $\mu^{C(i)} = \mu^{r(a,b)} = \{\}$
2: **for** each uml::Element E $\in$ M **do**
3:   **if** E isA uml::InstanceSpecification **then**
4:     **if** E.GetClassifiers $\neq \{\}$ **then**
5:       **for** each E.GetClassifiers C $\in$ M **do**
6:         $\langle E, C(E) \rangle \in \mu_{C(i)}$
7:       **end for**
8:     **else**
9:       $\langle$E,cora:Entity(E)$\rangle \in \mu_{C(i)}$
10:     **end if**
11:     **if** E isA uml::Relationship **then**
12:       source= a
13:       target= b
14:       $\langle E, E(a,b) \rangle \in \mu_{r(a,b)}$
15:     **end if**
16:   **end if**
17: **end for**
18: **return** $\mu^{\mathcal{A}} = \{\mu_{C(i)}, \mu_{r(a,b)}\}$

Algorithm 2: Construction of $\mu^{\mathcal{A}}$.

(11), we obtain (12) the source a and (13) target b of the relation, and state that $\langle E, E(a,b) \rangle \in \mu_{r(a,b)}$. Note that only stereotyped relations are captured. Once $\mu^{\mathcal{A}}$ and $\mu^{\mathcal{T}}$ are constructed, we can apply them to an UML model M, to automatically obtain $\mathcal{T}_{UML}$ and $\mathcal{A}_{UML}$. These artifacts can be reused by new designs, or exploited by external tools and services that are compatible with *ODrone* and CORA, thus effectively allowing the system designer to extend the ontology via UML.

5 CONCLUSIONS

Our approach tackled two main problems: the availability and integration of domain-specific ontologies into MBSE; and the capture and formalization of knowledge from the expert's design.

We have provided an end-to-end solution for the integration of formal vocabularies into MBSE tooling, effectively enabling the system designer to describe its system in terms of the ontology. This integration provides the context for other viewpoints and stakeholders to interact. We have formally defined the components a mapping from the terminology of an ontology, to UML and back should consider (encoded in OWL/UML format). This specification aims to ensure that the application of the mappings is consistent w.r.t. the domain-specific ontology. Furthermore, by clearly defining a system specification as a T-Box and a system instance as an A-Box, we separate these two aspects of system design, and we enable logic-based techniques (like reasoning or SAT solving) to evaluate and generate models for these artifacts.

As further work, we aim to enable the integration of different ontologies, different system design tools (e.g. safety, simulation, etc.), and to explore model generation for the obtained specifications.

REFERENCES

Atkinson, C. and Kiko, K. (2005). A detailed comparison of uml and owl. *None*.

Baader, F. (2003). Appendix: description logic terminology. *The Description logic handbook: Theory, implementation, and applications*, pages 485–495.

Baader, F., Calvanese, D., McGuinness, D., Patel-Schneider, P., Nardi, D., et al. (2003). *The description logic handbook: Theory, implementation and applications*. Cambridge university press.

Baader, F., Horrocks, I., Lutz, C., and Sattler, U. (2017). *Introduction to description logic*. Cambridge University Press.

Berardi, D., Calvanese, D., and De Giacomo, G. (2005). Reasoning on uml class diagrams. *Artificial intelligence*, 168(1-2):70–118.

Elasri, H. and Sekkaki, A. (2013). Semantic integration process of business components to support information system designers. *arXiv preprint arXiv:1302.1393*.

Hallberg, N., Jungert, E., and Pilemalm, S. (2014). Ontology for systems development. *International journal of software engineering and knowledge engineering*, 24(03):329–345.

Medinacelli, L. P., Noyrit, F., and Mraidha, C. (2022). Augmenting model-based systems engineering with knowledge. In *Proceedings of the 25th International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings*, pages 351–358.

Mezhuyev, V. (2014). Ontology based development of domain specific languages for systems engineering. In *2014 International Conference on Computer and Information Sciences (ICCOINS)*, pages 1–6. IEEE.

Mkhinini, M. M., Labbani-Narsis, O., and Nicolle, C. (2020). Combining uml and ontology: An exploratory survey. *Computer Science Review*, 35:100223.

Parreiras, F. S. (2011). Marrying model-driven engineering and ontology technologies: The twouse approach.

Vanherpen, K. e. a. (2016). Ontological reasoning for consistency in the design of cyber-physical systems. In *2016 1st International Workshop on Cyber-Physical Production Systems (CPPS)*, pages 1–8. IEEE.

Yang, L., Cormican, K., and Yu, M. (2019). Ontology-based systems engineering: A state-of-the-art review. *Computers in Industry*, 111:148–171.