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 SysML1, a UML2 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...
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 OWL3 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-

3https://www.w3.org/TR/owl2-overview/
vice, such as consistency or instance checking.

In our work we consider the description logics $\mathcal{ALCHI}^D$. The rationale behind this choice lies in that we aimed to at least provide $\mathcal{ALCH}$’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 $\mathcal{ALCHI}^D$.

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 $o_1, o_2, \ldots, o_n \in N_I$ then $\{o_1, o_2, \ldots, o_n\}$ is a concept. If $\phi_1$ and $\phi_2$ 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 $\top$ and $\bot$, respectively. We eliminate parentheses as usual.

**Axioms.** An axiom in $\mathcal{ALCHI}$ has one of the following forms: 1) $\phi \sqcup \psi$ (called concept inclusion axiom), where $\phi$ and $\psi$ 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) $\psi(a)$ (called concept membership axiom), where $\phi$ 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 $a, b \in N_I$ (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 $(T, A)$ for T-Box $T$ and A-Box $A$. 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 lowercase, 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 $O^c = \{C_1, C_2, \ldots, C_n\}$ is the set of concept names in ontology $O$, and $O^r = \{r_1, r_2, \ldots, r_m\}$ 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:

$O_d \rightarrow O_{up} = \{(IC, IR)\}$

where:

- $IC = \{(C_1^D, C_2^D, \ldots, C_n^D)\}$
- $IR = \{(r_1^D, r_2^D, \ldots, r_m^D)\}$

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

**Example 1 (Upper and Domain-Specific Ontologies).** Let $O_{CORA}$ and $O_{Drone}$ be two ontologies, with $O^c_{CORA} = \{Device, ActuatorDevice\}$, $O^c_{Drone} = \{SensorDevice, Camera, Motor, Propeller\}$, $O^r_{Drone} = \{hasPart, isConnectedTo\}$.

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

$O_{Drone} \rightarrow O_{CORA} = \{(\{SensorDevice, Device\}, \{Motor, ActuatorDevice\}\}, \{hasPart, part\}, \{isConnectedTo, interactsWith\}\}$

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

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

Where each $C_i^D, r_j^D \in IC$ with $n = |IC|$ and, each $r_j^D \in IR$ with $m = |IR|$.

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

$U_{O_{Drone}, O_{CORA}} = O_d \cup O_{up} \cup \{SensorDevice \subseteq Device, Motor \subseteq ActuatorDevice\} \cup \{\text{hasPart} \subseteq \text{part}^{-}, \text{isConnectedTo} \subseteq \text{interactsWith}\}$
**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 case-dependent. Moreover, there are specific sets of confinement. UML constructs, that can be specialized and/or objectified by the system designer to specify his model. We target these constructs to define a mapping between OWL and UML.

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

$$\sigma_{uml} = \{ U_1, U_2, \ldots, U_n \}$$

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 $O'$, and the set of role names $O_{rel}$.

We define the mapping:

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

with:

$$\mu_C = \{ (C_1, U_1), (C_2, U_2), \ldots, (C_m, U_m) \}$$

$$\mu_{OP} = \{ (r_1, U_1), (r_2, U_2), \ldots, (r_n, U_n) \}$$

$$\mu_{DP} = \{ (r_1, U_1), (r_2, U_2), \ldots, (r_p, U_p) \}$$

where:

1) $\forall (x, y) \in \mu_C \mid x \in O_{rel}, y \in \sigma_{uml}, (z, y) \notin \{ \mu_{OP}, \mu_{DP} \}$

2) $\forall (x, y) \in \mu_{OP} \mid x \in O, y \in \sigma_{uml}, (z, y) \notin \{ \mu_C, \mu_{DP} \}$

3) $\forall (x, y) \in \mu_{DP} \mid x \in O_{rel}, y \in \sigma_{uml}, (z, y) \notin \{ \mu_C, \mu_{OP} \}$

**Example 3 (Mapping $\mu^{O_{uml} \rightarrow O}$), where:**

$$\mu_C = \{ \langle \text{Device}, \text{uml} :: \text{Class} \rangle, \langle \text{Device}, \text{uml} :: \text{Component} \rangle \}$$

$$\mu_{OP} = \{ \langle \text{hasPart}, \text{uml} :: \text{Element} :: \text{ownedElement} \rangle, \langle \text{isConnectedTo}, \text{uml} :: \text{Association} \rangle \}$$

$$\mu_{DP} = \{ \}$$

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 $\mu^{O_{uml} \rightarrow O}$ does not map names in $O$ to actual UML models elements, but to its meta-language. The mapping $\mu^{O_{uml} \rightarrow O}$ 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 $\mu^{O_{uml} \rightarrow O}$, 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 $\mu^{O_{uml} \rightarrow M}$).** Let $M$ be a UML model, and let

$$\sigma_M = \{ M_1, M_2, \ldots, M_n \}$$

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

$$\mu^{O_{uml} \rightarrow M} = \{ \mu_{\subseteq}, \mu_{\equiv}, \mu_{OP}, \mu_{DP} \}$$

with:

$$\mu_{\subseteq} = \{ (M_1, C_1), (M_2, C_2), \ldots, (M_n, C_n) \}$$

$$\mu_{\equiv} = \{ (M_1, C_1), (M_2, C_2), \ldots, (M_n, C_n) \}$$

$$\mu_{OP} = \{ (M_1, r_1), (M_2, r_2), \ldots, (M_n, r_n) \}$$

$$\mu_{DP} = \{ (M_1, C(i)_1), (M_2, C(i)_1), \ldots, (M_m, C(i)_m) \}$$

$$\mu_{r(a,b)} = \{ (M_1, r(a, b)_1), (M_2, r(a, b)_2), \ldots, (M_n, r(a, b)_n) \}$$

with $i, j, k, l, m, n \geq 0$. Where each $C_i$ is a (possibly complex) concept in $\mathcal{ACHF}$, each $r_i$ is a role in...
O, and where \(\langle M_i, x \rangle\) and \(\langle M_i, y \rangle\) can not occur, if \(x \neq y\).

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

\[
Q_{\text{UML}} = M^{\mu_{\text{UML}} \rightarrow O}
\]

, recursively defined by:

\[
\begin{align*}
\forall (A, B) \in \mu &\Leftrightarrow \{ A \sqsubseteq B \} \in Q_{\text{UML}} \\
\forall (A, B) \in \mu &= \{ A \equiv B \} \in Q_{\text{UML}} \\
\forall (A, B) \in \mu &\equiv \{ A \equiv B \} \in Q_{\text{UML}} \\
\forall (A, B) \in \mu &\equiv \{ A \equiv B \} \in Q_{\text{UML}} \\
\forall (A, C) \in \mu (i) &\equiv \{ C(i) \} \in Q_{\text{UML}} \\
\forall (A, r(a, b)) \in \mu &\equiv \{ r(a, b) \} \in Q_{\text{UML}}
\end{align*}
\]

**Example 4** (Definition and Application of \(\mu_{\text{UML}} \rightarrow O\)). Consider a UML model \(M\), a simplified version of the model in figure 1, containing the following elements:

\[
\sigma_M = \{ \text{DroneSystem1, BatteryType1, MotorType1, PropellerType1, } \}
\]

\[
\begin{align*}
\mu &= \{ \text{PropellerType1, Propeller, MotorType1, Motor, } \\
\mu &= \{ \text{DroneSystem1, Device}, \text{DroneSystem1, PropellerType1} \}
\end{align*}
\]

\[
\begin{align*}
\mu &= \{ \text{PropellerType1, MotorType1}, \text{BatteryType1} \}
\end{align*}
\]

Then, the application of \(\mu_{\text{UML}} \rightarrow O\) to model \(M\) yields:

\[
Q_{\text{UML}} = \{ \\
\text{PropellerType1, MotorType1, DroneSystem1, } \\
\text{hasPart, MotorType1, } \\
\text{connectedTo, hasPart, MotorType1} \}
\]

**Definition 7** (A System Specification T-Box). Given an UML model \(M\) and the mapping \(\mu_{\text{UML}} \rightarrow O\), the System Specification T-Box \(T_{\text{UML}}\) is the result of applying the restricted mapping:

\[
\mu_T = \{ \mu_C, \mu_E, \mu_{\text{OP}}, \mu_{\text{DP}} \}
\]

**Example 5** (System Specification T-Box). Given the UML model \(M'\) and the mapping \(\mu_T\), the application of \(\mu_T\) to \(M\) yields:

\[
T_{\text{UML}} = \{ \\
\text{PropellerType1, MotorType1, } \\
\text{DroneSystem1, Device}, \text{DroneSystem1, PropellerType1} \}
\]

**Definition 8** (A UML System Instance). Given an UML model \(M\) and the mapping \(\mu_{\text{UML}} \rightarrow O\), a system instance A-Box \(A_{\text{UML}}\) is the result of applying the restricted mapping:

\[
\mu_A = \{ \mu_C(i, j), \mu_E(a, b) \}
\]

**Example 6** (System Instance). Given the UML model \(M\) and the mapping \(\mu_A\), the application of \(\mu_A\) to \(M\) yields:

\[
A_{\text{UML}} = \{ \text{MotorType1, MotorType1, } \\
\text{DroneSystem1, PropellerType1} \}
\]

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

\[
Q_{\text{UML}} \cup T_{\text{UML}} \models T
\]

**Definition 10** (System Specification Model). Given an ontology \(O\), an UML model \(M\), a mapping \(\mu_{\text{UML}} \rightarrow O\), and a consistent system specification \(T_{\text{UML}}\), a system specification model for the consistent specification \(T_{\text{UML}}\) is an A-Box \(A_{\text{UML}}\) s.t.

\[
Q_{\text{UML}} \cup T_{\text{UML}} \cup A_{\text{UML}} \models T
\]

**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 tool).
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: \( \text{CORA.owl} = O_1 \) and \( \text{ODrone.owl} = O_2 \). \( \text{ODrone.owl} \) 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 \( \text{Union}_O : O_2 \rightarrow O_1 \) (see section 3). The union ontology \( \text{Union}_O \) consistently integrates the domain-specific ontology ODrone into the upper and standardized ontology CORA. Thus providing a standardized domain-specific ontology \( \text{Union}_O \). Task 2 maps concepts and roles from \( \text{Union}_O \) to a UML profile thanks to the mapping specification \( \mu^{O \rightarrow UML} \), 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 \( \mu^{UML \rightarrow O} \) to a specific UML model \( M \). Once the annotated model is ready, the application of the “inverse” mapping \( \mu^{UML \rightarrow O} \) takes place in task 4. This process yields \( O_{UML} \), an ODrone/OWL compliant representation of the UML system, from where 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 Diagram 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 \( \mu^T \) with which we can obtain a system specification \( \mathcal{S}_{\text{UML}} \). Whereas algorithm 2, constructs the mapping \( \mu^A \), which can be applied to a UML model \( M \) to obtain a system instance \( \mathcal{A}_{\text{UML}} \).

These two algorithms are presented next.

Recall:

\[
\mu^T = \{ \mu_e, \mu_s, \mu_{OP}, \mu_{DP} \}
\]

In algorithm 1 we first (1) initialize the sets \( \mu_e, \mu_s, \mu_{OP}, \mu_{DP} \) 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 \( \langle \text{uml}::\text{Class}, \text{owl}::\text{Thing} \rangle \) to \( \mu_e \), 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 \( \mathcal{CCD}_E = \{ \} \) (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 concept complex 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 \( \langle E, \mathcal{CCD}_E \rangle \) belongs to \( \mu_s \), effectively providing a concept complex definition for \( E \), in terms of ODrone. Finally, in (29) a subsumption relation is added to \( \mu_{OP} \) 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 \( \mu^A \). Recall:

\[
\mu^A = \{ \mu_{C(i)}, \mu_{R(i,a,b)} \}
\]

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

**Input:** \( M, \mu^O_{\text{UML}} \)

**Output:** \( \mu^T \)

1: \( \mu_e = \mu_s = \mu_{OP} = \mu_{DP} = \{ \} \)

2: for each uml::Element \( E \in M \) do

3: if \( E \) isA uml::Class then

4: \( \langle E, \text{owl}::\text{Thing} \rangle \in \mu_e \)

5: if \( E \) specializesOf \( B \in M \) then

6: \( \langle E, B \rangle \in \mu_e \)

7: end if

8: for each E.AppliedStereotypes \( S \in M \) do

9: \( \langle E, S \rangle \in \mu_e \)

10: end for

11: \( \mathcal{CCD}_E = \{ \} \)

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: \( S = \text{odrone}:\text{hasPart} \)

16: \( \mathcal{CCD}_E = \mathcal{CCD}_E \cap \{ S, b \} \)

17: end if

18: end for

19: for each E.getOwnedElements \( E \in M \) do

20: if \( OE \) isA uml::CompositeAggregation then

21: \( b = OE\).getType

22: \( S = \text{odrone}:\text{hasPart} \)

23: \( \mathcal{CCD}_E = \mathcal{CCD}_E \cap \{ S, b \} \)

24: end if

25: end for

26: \( \langle E, \mathcal{CCD}_E \rangle \in \mu_e \)

27: end if

28: end for

29: if \( E \) isA uml::Relationship and \( E\).AppliedStereotypes \( \neq \emptyset \) then

30: \( S = R\).AppliedStereotype

31: \( \langle E, S \rangle \in \mu_{OP} \)

32: end if

33: return \( \mu^T = \{ \mu_e, \mu_s, \mu_{OP}, \mu_{DP} \} \)

**Algorithm 1:** Construction of \( \mu^T \).

In algorithm 2 we first (1) initialize the sets \( \mu_{C(i)}, \mu_{R(i,a,b)} \) 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. \( \text{cora}::\text{Entity} \). If it is the case the element \( E \) is an instance specification of a relationship
Algorithm 2: Construction of $\mu^A$.

(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 $\mu^A$ and $\mu^T$ 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


