


Implementing OntoUML Models with OntoObject-Z Specifications: A Proof of Concept Relying on a Partial Ontology for VLANs

Mohamed Bettaz ^a

Faculty of Information Technology, Czech Technical University in Prague, Thakurova 9, Prague, Czech Republic


Keywords: Ontology, UML, UFO, OntoUML, Object-Z, OntoObject-Z, Descriptive Language, Metamodel, EBNF, VLAN.

Abstract: OntoObject-Z is a descriptive language inspired by OntoUML. Just as OntoUML is a profile for the Unified Modeling Language (UML), OntoObject-Z is an extension of the Object-Z notation. The objective of this article is threefold. We first define a metamodel for OntoObject-Z and an EBNF-like notation formalizing the syntax of OntoObject-Z specifications. Second, we construct a partial ontology for Virtual Local Area Networks (VLANs) and describe it by OntoUML models. Third, we implement our OntoUML models with OntoObject-Z specifications. The OntoObject-Z metamodel is expressed in OntoUML and the proposed EBNF rules are based on OntoUML concepts. Thanks to this, each syntactically correct OntoObject-Z specification corresponds de facto to a correct implementation of an OntoUML model.

1 INTRODUCTION

OntoObject-Z (Bettaz and Maouche, 2023a) is a descriptive language inspired by OntoUML, an ontologically well-defined language used for conceptual modelling (Guizzardi, 2005). Just as OntoUML is a UML profile, OntoObject-Z is an extension to the Object-Z notation (Smith, 2000). Our interest in UML and Object-Z is motivated by the fact that, despite their relative longevity, both continue to attract interest from the academic and industrial communities. In addition, these two languages appear prominently in several undergraduate and graduate programs. Finally, both are equipped with modelling and verification tools (TOZE for Object-Z, and OpenPonk for UML and OntoUML). TOZE was used to edit the OntoObject-Z specifications presented in Section 7, and OpenPonk to edit and verify the OntoObject-Z metamodel presented in Section 5 and the OntoUML models presented in Section 6. Our interest in OntoUML and OntoObject-Z is motivated by their status of prescriptive languages acquired thanks to their anchoring in ontologies (cf. Section 4 and Section 5). For a good discussion on the prescriptiveness and descriptiveness of models and ontologies, the kind reader may consult the excellent work of (Abmann et al., 2006). The basic ingredients of OntoObject-

Z were introduced in (Bettaz and Maouche, 2023a) and (Bettaz and Maouche, 2023b), where it was informally shown how to map some of the basic OntoUML constructs into OntoObject-Z. In this paper, we address one of the main points announced as “future work” in (Bettaz and Maouche, 2023a), which is to provide OntoObject-Z with a metamodel, aiming to be the very first step towards the formalization of this language. Second, we construct a partial ontology for VLANs and describe it with OntoUML models. Third, we implement our OntoUML models with OntoObject-Z specifications. The OntoObject-Z metamodel is expressed in OntoUML and the proposed EBNF rules are based on OntoUML concepts. Thanks to this, each syntactically correct OntoObject-Z specification corresponds de facto to a correct implementation of an OntoUML model. Equipped with a relatively “diverse” set of OntoUML constructs, we believe that our VLANs’ ontology can serve as a proof of concept for such an implementation. A first motivation behind the idea of implementing OntoUML models with OntoObject-Z specifications is to describe ontologies in a formal descriptive language. A second motivation is to address most of the phases of the Software Development Life Cycle (SDLC) in a language equipped with a refinement method. Indeed OntoObject-Z, based on Object-Z, “benefits” from such a method. In (Bettaz and Maouche, 2023b), we define a refinement approach that can be exploited in

^a  <https://orcid.org/0000-0003-1346-0244>

this sense. A third motivation is to express specifications and constraints on them in a single language. Indeed like UML, OntoUML uses OCL (Object Constraint Language) to express constraints on domain descriptions. The construction of an ontology for VLANs is motivated by our interest in domain knowledge (Bettaz, 2024), a concept known from the software engineering discipline. Indeed, to develop an application or system for a domain, you need a certain expertise (knowledge) of the domain. In this context, to develop “high quality” protocol-specific applications, you need in-depth knowledge of protocol-based VLANs (not just port-based or address-based VLANs). In this sense VLANs can be considered as a domain in their own right. More details about the motivation behind our choice of building an ontology for VLANs are given in Section 6.

The rest of this paper is organized as follows. Section 2 presents our research method. In Section 3 we discuss some related works. Section 4 gives succinct information about the Unified Foundation Ontology (UFO) and OntoUML. In Section 5 we present our metamodel for OntoObject-Z and the EBNF rules formalizing the syntax of OntoObject-Z specifications. Section 6 is devoted to the ontology for VLANs and its description by OntoUML. The implementation of the OntoUML models describing our VLANs’ ontology with OntoObject-Z specifications is presented in Section 7. Concluding remarks and future work are presented in Section 8.

2 RESEARCH METHOD

This paper uses a formal approach (Roggenbach et al., 2022) to provide OntoObject-Z with a metamodel and OntoObject-Z specifications with a formal notation describing their syntax. The OntoObject-Z metamodel is expressed in OntoUML and the formal notation (describing the syntax of OntoObject-Z specifications) is expressed by EBNF rules based on OntoUML concepts. Thanks to this, each syntactically correct OntoObject-Z specification corresponds de facto to a correct implementation of an OntoUML model. For the construction of our VLANs’ ontology we use an analytical method and a “reverse-engineering” approach relying on material from specialized literature such as, (Comer, 2015), (Odom, 2016), (Seifert and Edwards, 2008), and sometimes on our long-life experience in teaching and research.

3 RELATED WORKS

Mapping between UML and Object-Z dates back to the work of (Kim and Carrington, 2000). An institution based on ideas from this work gives a formal syntax and semantics to Object-Z (Baumeister et al., 2015). The work in (Bettaz and Maouche, 2017) shows how a “combination” of UML and OCL can be mapped into Object-Z. The objective was to show that Object-Z can serve as an implementation language for “UML + OCL”. Indeed, thanks to its predicate part, Object-Z can implement the constraints expressed in OCL. From an SDLC perspective, this leads to the adoption of an “evolutionary” approach instead of a “discrete” approach. This applies mainly in the context of Model Driven Requirements Engineering (MDRE) (Assar, 2014). In an evolutionary approach, most of the SDLC phases are processed using a single language, where we proceed through stepwise refinement. For this, a refinement idea based on the transformation of inheritance into delegation can be exploited for OntoObject-Z (Bettaz and Maouche, 2023b). In a discrete approach a different language is used to process each phase, which raises the problem of “correct” mapping of specifications expressed in various languages.

4 UFO AND OntoUML

OntoUML is a language for conceptual modelling, conceived as a profile of UML based on UFO. Reporting on OntoUML and UFO is out of the scope of this paper; the reader interested in more detail is kindly directed to the rich bibliography treating on both subjects - cf. for instance (Guizzardi et al., 2021) and (Pergl, 2023). In a nutshell, UFO classifies “entities” into two broad “categories” named sortals and non-sortals. Instances (individuals) of sortals (types) always have one and the same identity principle, and each instance must have exactly one identity. Instances of non-sortals can have different identity principles. Indeed, non-sortals are abstract classes. Sortals and non-sortals can be either rigid or anti-rigid. Rigidity is a qualifier for entities that are unable to adapt to change, while anti-rigidity is a qualifier for entities that do. The concepts of rigidity and anti-rigidity are formalized in modal logic; cf. for instance (Guizzardi et al., 2021) and (Pergl, 2023). Examples of rigid sortals, anti-rigid sortals, rigid non-sortal, and anti-rigid non-sortal are given in Section 6. OntoUML uses sortals and non-sortals in their rigid and anti-rigid form as stereotypes for its classes (in the sense of UML classes). Moreover, in OntoUML, as-

sociations are also stereotyped. These stereotypes and others (cf. Section 5) bring a noticeable change in the semantics of UML models. A stereotype is placed between `<<` and `>>`. Stereotype (Kind), for instance, is written as `<<Kind>>`.

5 OntoObject-Z

The basic ingredients of OntoObject-Z were introduced in (Bettaz and Maouche, 2023a). In this section, we present a metamodel for OntoObject-Z (cf. Figure 1) and an EBNF-like notation formalizing the syntax of OntoObject-Z specifications.

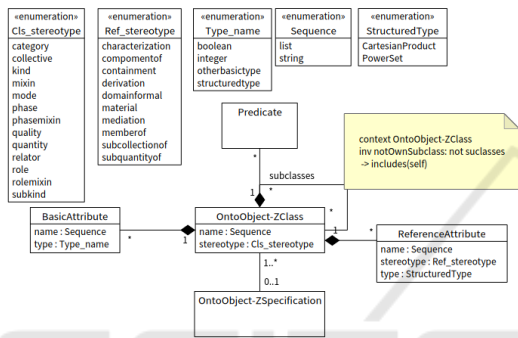


Figure 1: An OntoUML metamodel for OntoObject-Z.

Since OntoObject-Z borrows stereotypes from OntoUML, it is quite “natural” to build the very first OntoObject-Z metamodel in OntoUML. Another reason is that OntoUML is supported by a tool (OpenPonk) allowing to verify OntoUML models. For sake of readability, we will write between braces the names used in the different figures when they are mentioned in the text. Our metamodel (cf. Figure 1) shows that an OntoObject-Z class (OntoObject-ZClass) is a “whole” having a name and a stereotype. It consists of three “parts”: a basic attribute part (BasicAttribute) representing its basic attributes, a reference attribute part (ReferenceAttribute) representing its reference attributes, and a predicate part (Predicate) representing its predicates. A basic attribute has name and a type. A reference attribute has a name, a type and a stereotype. The types of the various attributes relating to the whole and to the various parts are defined by enumeration type declarations, which are nothing more than UFO stereotypes used in OntoUML. A formal definition for basic and reference attributes (including the significance of stereotyping for reference attributes) is given in (Bettaz and Maouche, 2023a). The OCL constraint associated with the association (subclasses) states that an OntoObject-Z class cannot be its own subclass. Similar OCL expressions can

be formulated to specify constraints on relationships between the various stereotypes defined for class stereotypes (Cls_stereotype). Also, the OntoObject-Z metamodel defines an OntoObject-Z specification (OntoObject-ZSpecification) as a collection of several OntoObject-Z classes.

We define the formal syntax for OntoObjectZ specifications with the following EBNF-like rules.

```

<OntoObject-Z_specification> ::= {OntoObject-Z_class}+
<OntoObject-Z_class> ::= <header> <state>
<header> ::= <header_of_sortal_class> |
<header_of_nonsortal_class>
<header_of_sortal_class> ::=
<<<cls_stereotype>>>
<cls_name> [{<cls_name>}]
<header_of_nonsortal_class> ::=
<<<cls_stereotype>>> <cls_name> : abstract
<cls_stereotype> ::= category | collective | kind |
mixin | mode | phase | phasemixin | quality | quantity
| relator | role | rolemixin | subkind
<state> ::= [<declaration>] [<predicate>]
<declaration> ::= {<basic_attribute> |
<reference_attribute>}
<basic_attribute> ::=
<<<cls_stereotype>>>
<cls_name> :
<bas_attribute_name> : <type_name>
<type_name> ::= boolean | integer | ...
<ref_attribute> ::=
<<<cls_stereotype>>>
<cls_name> :
<<<ref_stereotype>>> <ref_attribute_name> :
ℙ <<<ref_stereotype>>> <cls_name>
<<<ref_stereotype>>> ::= characterization | componentof
| containment | derivation | domaininformal | material |
mediation | memberof | subcollectionof | subquantityof
<predicate> ::=
{<multiplicity_predicate> |
<navigability_predicate> |
<is_whole_for_predicate> | <is_part_of_predicate>
| isDisjoint | isComplete}
<multiplicity_predicate> ::=
# <ref_attribute_name> <rel_operator> <value>
<rel_operator> ::= >= | <= | = | < | > | !=
<value> ::= 0 | 1 | 2 | ...
<is_whole_for_predicate> ::= <cls_name> isWholeFor <cls_name>
<is_part_of_predicate> ::= <cls_name> isPartOf <cls_name>
<navigability_predicate> ::= = ∇ <var_name> :
<cls_name> • self in <var_name> • <cls_name>
    
```

The formal definition for the predicates (isWhole-

For) and (isPartOf) can be defined using the predicate logic in the same way as (isDisjoint) and (isComplete) were defined in (Bettaz and Maouche, 2023a).

6 DESCRIPTION OF THE VLANS' ONTOLOGY

In this section we present our ontology for VLANs and its description with OntoUML. Describing VLANs in OntoUML allows to have a shared conceptualization not only for various (hardware / software) entities composing a VLAN but also for the knowledge (implicitly) defined by their interrelationships. So, while it is probably obvious to list all the concepts revolving around VLANs such as broadcast domains, collision domains, LANs' generations, computing devices, transmission media, virtual LAN switches, port-based VLANs, address-based VLANs, Layer 3-based VLANs, native VLANs, VLANs using separate physical links, VLANs using trunks, Dot1q protocol, routing between VLANs, VLANs using routers or Layer 3 switches, bridged LANs, bridged LANs containing cycles, spanning trees, spanning tree protocols, forwarding tables, forwarding algorithms, blocked ports, root ports, designated ports, disconnected bridges, designated bridges and so forth, nothing is less obvious than establishing all the interrelationships in a concise, complete and consistent manner. For sake of readability, our model for VLANs' ontology is decomposed into four (ontology) "fragments". The first fragment presents various generations of LANs. The second fragment describes the concept of VLAN switch and various types of VLANs (port-based, address-based, and Layer 3-based). The third fragment addresses the notion of routing between VLANs. The fourth fragment introduces the concept of bridged LANs, the notions of spanning tree, and Spanning Tree Protocol (STP) as well. Classes that are "replicated" serve to "glue" the various fragments.

6.1 LAN Generations

The ontology fragment depicted in Figure 2 should be read as follows. A LAN (<<Kind>> LAN) is seen as a kind of the category of computer networks (<<Category>> Computer Network). Kinds (stereotype <<Kind>>) are rigid sortals, while categories (stereotype <<Category>>) are rigid non-sortals. Non-sortals in UFO (and in OntoUML) are abstract classes (in the sense of object-oriented programming), and they are written in italic. The rigid sortal (<<Mode>> Broadcast Domain) expresses

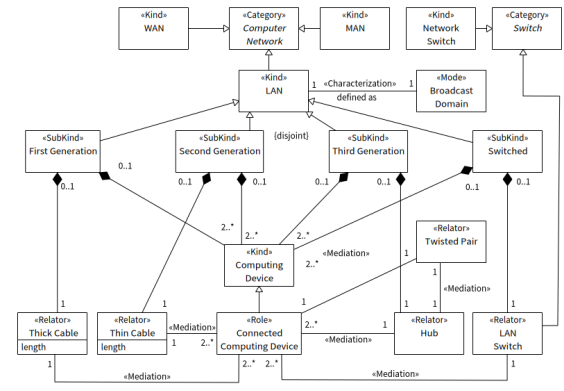


Figure 2: LAN Generations: an OntoUML representation.

the fact that a LAN is defined as a broadcast domain. Stereotype <<Mode>> is used to emphasize the importance of a given characteristic of the "bearer" (<<Kind>> LAN) in our case. The association between a mode and a bearer should be "decorated" by the stereotype (<<characterization>>). The various LAN generations are all seen as LANs, and are de facto also broadcast domains, since they all inherit the relationship (<< characterization >> defined as). As expected, the various generations of LANs are distinguished just by the type of the used transmission medium (<<Relator>> Thick Cable), (<<Relator>> Thin Cable), etc. The relationships ending by a diamond describe a "whole-part" relationship. A "filled" diamond expresses the fact that a "part" is not shareable (a thick cable is not shared by two different LANs, and so on.) The multiplicities written next to the diamonds express the fact that the "whole" is not mandatory for the "part" (while a first generation LAN necessitates two or more computing devices, a first generation LAN is not mandatory for a computing device). The various kinds of transmission media are seen as relators (stereotype <<Relator>>). A thick cable, for instance, relates (mediates) two or more connected computing devices. A connected computing device (<<Role>> Connected Computing Device) is seen as a role played by a computing device (<<Kind>> Computing Device).

6.2 VLAN Types

The ontology fragment depicted in Figure 3 should be read as follows. A VLAN (<<Subkind>> VLAN) is a switched LAN (<<Subkind>> Switched), which is itself a LAN (<<Kind>> LAN). As a LAN, a VLAN is also defined as a broadcast domain. A VLAN can be created or removed at any time. Thus, we have two phases: (<<Phase>> Cre-

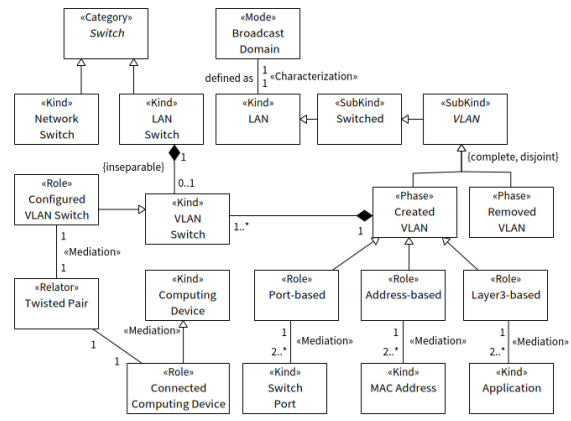


Figure 3: VLAN Types.

ated VLAN) and («Phase» Removed VLAN). («Subkind» VLAN) is an abstract class because these two phases define a generalization set that is complete and disjoint. This is expressed by the meta-attribute {complete, disjoint}. A VLAN in the first phase («Phase» Created VLAN) consists of one or more VLAN switches («Kind» VLAN Switch), where a VLAN switch is seen as an “inseparable part” of a LAN switch (the meta-attribute {inseparable} is used for this purpose). Inseparable here means that an instance of («Kind» VLAN Switch) cannot exist without the same instance of («Kind» LAN Switch). The various types of VLAN («Role» Port-based), («Role» Address-based), and («Role» Layer3-based) are modelled as roles played by a created VLAN. As expected, a port-based VLAN is associated with two or more ports, an address-based VLAN is associated with two or more MAC addresses, and a Layer3-based VLAN is associated with two or more network applications. A configured VLAN switch («Role» Configured VLAN Switch) and a connected computing device («Role» Connected Computing Device) are related by twisted pairs («Relator» Twisted Pair).

6.3 Routing Between VLANs

The ontology fragment depicted in Figure 4 should be read as follows. A switched VLAN («SubKind» VLAN) using a trunk («Role» Using a Trunk) and a switched VLAN using separate physical links («Role» Using Separate Links) can be seen as two different roles played by a VLAN («SubKind» VLAN). In such a situation the router plays also two roles («Role» Connected to Trunk) and («Role» Connected to Separate Links). The relators («Relator» Trunk) and

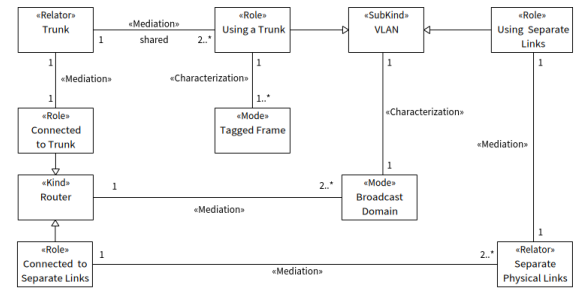


Figure 4: Routing between VLAN.

(«Relator» Separate Physical Links) mediate, respectively, both of these material relations. As we know, when a VLAN is using a trunk, the frames are tagged before crossing the router. This is represented by («Mode» Tagged Frame).

6.4 Bridged LAN

The ontology fragment depicted in Figure 5 should be read as follows. A bridged LAN («Kind»

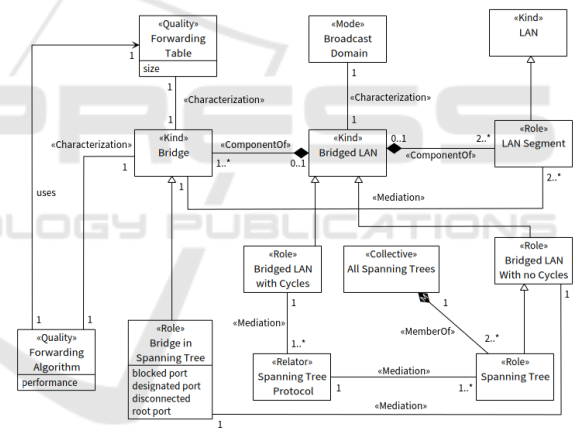


Figure 5: Bridged LAN.

Bridged LAN) consists of a bridge («Kind» Bridge) and two or more LAN segments («Role» LAN Segment). The bridge is “characterized” by a forwarding table («Quality» Forwarding Table), with a distinguished attribute defining its size, and a forwarding algorithm («Quality» Forwarding Algorithm) with a distinguished attribute defining its performance. A bridged LAN, as a LAN, is also seen as a broadcast domain («Mode» Broadcast Domain). Both bridged LAN with cycles («Role» Bridged LAN with Cycles) and bridged LAN with no cycles («Role» Bridged LAN With no Cycles) are seen as roles of a bridged LAN. The spanning tree protocol («Relator» Spanning Tree Protocol) is used to associate a spanning tree («Role»

Spanning Tree) with a bridged LAN with cycles. All spanning trees form a collective (<<Collective>> All Spanning Trees). A bridge in a spanning tree (<<Role>> Bridge in Spanning Tree) possesses one root port, one designated port, and can have one or more of its ports blocked. In case that all of the ports are blocked, the bridge is merely disconnected. These are modelled as attributes of (<<Role>> Bridge in Spanning Tree).

7 IMPLEMENTATION OF THE VLANS' ONTOLOGY

For lack of space, only the implementation of the first fragment is presented in this Section. For sake of readability, the implementation of our first ontology fragment is represented by several pieces of OntoObject-Z specifications (cf. Figures 6, 7, 8, 9, and 10).

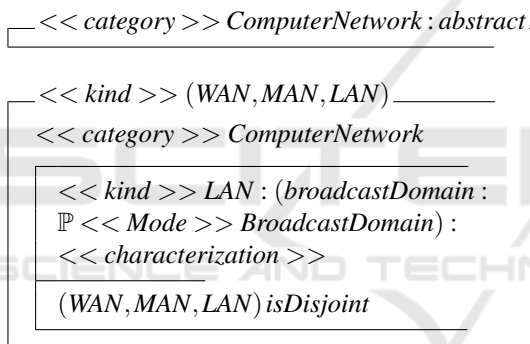


Figure 6: Networks kinds.

The piece of OntoObject-Z specification depicted in Figure 6 describes an OntoObject-Z abstract class (<<category>> ComputerNetwork: abstract) implementing the OntoUML abstract class of the same name, and an OntoObject-Z class (<<kind>> (WAN, MAN, LAN)) implementing the three OntoUML classes of the same name (cf. Subsection 6.1).

OntoObject-Z “enriches” Object-Z with a new notation that consists in regrouping the specification of two or more “child” classes together with their “parent” class. The motivation behind this, is to write concise and more readable specifications. Next, our piece of OntoObject-Z specification declares a reference attribute (broadcastDomain) for (<<kind>> LAN). This reference attribute has the type (\mathbb{P} BroadcastDomain) and the stereotype (<<characterization>>). (<<Mode>>) is the stereotype of the class (BroadcastDomain). The predicate part of our piece of OntoObject-Z specification

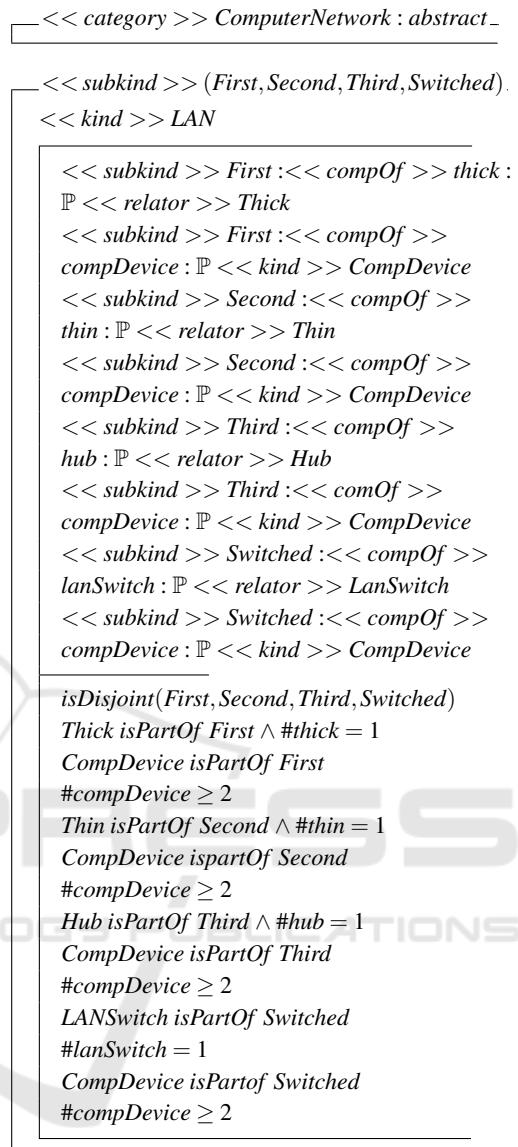


Figure 7: LAN generations.

consists of one predicate stating that the three subclasses (WAN, MAN, and LAN) are not overlapping. A formal specification of (isDisjoint), is given in (Betzaz and Maouche, 2023a).

The piece of OntoObject-Z specification depicted in Figure 7 defines an OntoObject-Z class (<<subkind>> (First, Second, Third, Switched)) regrouping three “children” of (<<Kind>> LAN) implementing, respectively, the four OntoUML classes (<<SubKind>> First Generation), (<<SubKind>> Second Generation), (<<SubKind>> Third Generation), and (<<SubKind>> Switched). The declaration part of this piece of specification defines the attributes

for each “child” as follows. The declaration of each reference attribute (stereotype, name, and type) is indexed by the name of the “child” (stereotype, name). For sake of readability the declaration of the attributes are regrouped by “child”. For instance, the “child” ($\ll\text{subkind}\gg$ Switched) has two attributes:

- i) a first attribute (thick) with stereotype ($\ll\text{CompOf}\gg$) and type ($\mathbb{P}\ll\text{relator}\gg$ Thick). The index is ($\ll\text{subkind}\gg$ First).
- ii) a second attribute (compDevice) with stereotype ($\ll\text{CompOf}\gg$) and type ($\mathbb{P}\ll\text{kind}\gg$ CompDevice). The index is the same, i.e., ($\ll\text{subkind}\gg$ First). The predicate part of our piece of specification (Figure 7) states several constraints that might be commented as follows. The meaning of the predicate ($\text{isDisjoint}(\text{First}, \text{Second}, \text{Third}, \text{Switched})$) states that the set-theoretic models of the classes (First, Second, Third, and Switched) as not overlapping. ($\text{Thick isPartOf First}$), a notation for ($\text{isPartOf}(\text{Thick}, \text{First})$), expresses the fact that in the whole-part relationship between the classes (ThickCable) and (FirstGeneration), (ThickCable) is the part and (FirstGeneration) is the whole. ($\#\text{thick} = 1$) states that a thick cable is a mandatory part for a first generation LAN.

The piece of specification depicted in Figure 8 defines an OntoObject-Z class ($\ll\text{kind}\gg$ CompDevice) implementing the OntoUML class (Computing Device). The declaration part defines four reference attributes that all are stereotyped as ($\ll\text{compOf}\gg$). Each attribute is typed by the power-set of the corresponding whole (i.e., First, Second, Third, and Switched). The predicate part can be commented as follows. ($\text{First isWholeFor CompDevice}$) is an expression for ($\text{isWholeFor}(\text{First}, \text{CompDevice})$) stating that a first generation LAN is a whole for each computing device connected to it. ($\#\text{first} = 0 \vee \#\text{first} = 1$) states that a first generation LAN is not mandatory for a computing device that could be connected to it. The predicate ($\forall f : \text{first} \bullet \text{self} \in f.\text{compDevice}$) defines the navigability of the association relating (First) and (compDevice).

The piece of specification depicted in Figure 9 defines an OntoObject-Z class ($\ll\text{role}\gg$ ConnecCompDevice) implementing the OntoUML class (Connected Computing Device).

The declaration part defines four reference attributes that all are stereotyped as ($\ll\text{mediation}\gg$). Each attribute is typed by the power-set of the referenced class. The predicate part can be commented as follows. ($\#\text{twisted} = 1$) means that a twisted pair is mandatory for a computing device connected to a LAN. The predicate ($\forall t : \text{twisted} \bullet \text{self} \in$

```

<< kind >> CompDevice
-----
<< compOf >> first :  $\mathbb{P}$ First
<< compOf >> second :  $\mathbb{P}$ Second
<< compOf >> third :  $\mathbb{P}$ Third
<< compOf >> switched :  $\mathbb{P}$ Switched

First isWholeFor CompDevice
#first = 0  $\vee$  #first = 1
 $\forall f : \text{first} \bullet \text{self} \in f.\text{compDevice}$ 
Second isWholeFor ComDevice
#second = 0  $\vee$  #second = 1
 $\forall s : \text{second} \bullet \text{self} \in s.\text{compDevice}$ 
Third isWholeFor CompDevice
#third = 0  $\vee$  #third = 1
 $\forall t : \text{third} \bullet \text{self} \in t.\text{compDevice}$ 
Switched isWholeFor CompDevice
#switched = 0  $\vee$  #switched = 1
 $\forall s : \text{switched} \bullet \text{self} \in s.\text{compDevice}$ 
    
```

Figure 8: Computing device.

```

<< role >> ConnecCompDevice
-----
ComputingDevice : << kind >>
-----
<< mediation >> twisted :  $\mathbb{P}$ Twisted
<< mediation >> thick :  $\mathbb{P}$ Thick
<< mediation >> thin :  $\mathbb{P}$ Thin
<< mediation >> hub :  $\mathbb{P}$ Hub
<< mediation >> lanSwitch :  $\mathbb{P}$ LANSwitch

#twisted = 1
 $\forall t : \text{twisted} \bullet \text{self} \in t.\text{connecCompDevice}$ 
#thick = 1
 $\forall t : \text{thick} \bullet \text{self} \in t.\text{connectCompDevice}$ 
#thin = 1
 $\forall t : \text{thin} \bullet \text{self} \in t.\text{connectCompDevice}$ 
#hub = 1
 $\forall h : \text{hub} \bullet \text{self} \in h.\text{connectCompDevice}$ 
#lanSwitch = 1
 $\forall l : \text{lanSwitch} \bullet \text{self} \in l.\text{connectCompDevice}$ 
    
```

Figure 9: Connected computing device.

```

<< category >> Switch
-----
<< kind >> NetSwitch, << relator >> LANSwitch.
<< category >> Switch
    
```

Figure 10: Category of switches.

$t.\text{connecCompDevice}$) defines the navigability of the association relating a connected computing device to

a twisted pair.

The piece of specification depicted in Figure 10 defines two OntoObject-Z classes (<<kind>> NetSwitch) and (<<relator>> LANSwitch) implementing the OntoUML abstract class (Switch).

8 CONCLUSIONS

The first result of this contribution consists in defining a metamodel for OntoObject-Z and formalizing the syntax of the specifications written in this language. This paves the way for a “sound” formalization of this promising language.

Building an ontology for VLANs and describing it in OntoUML is a second result of this work.

The third result is the implementation of our OntoUML models with OntoObject-Z specifications. The motivation behind the idea of implementing OntoUML models with OntoObject-Z specifications is given in Section 1 and Section 3. It is important to emphasize here that the approach used to build the metamodel and to write the EBNF rules makes it possible to achieve an implementation that is correct by construction.

Providing formal semantics for the implementation of OntoUML models with OntoObject-Z specifications can be approached in two ways. Using an OntoUML metamodel expressed in OntoUML and writing the rules mapping this metamodel into the OntoUML metamodel of Object-Z, or providing both OntoUML and OntoObject-Z with metamodels expressed in OntoObject-Z and writing the rules for mapping the first metamodel into the second one. This task is left for future work.

Another more elaborated future work is to provide OntoUML and OntoObject-Z with appropriate institutions (Diaconescu, 2023), (Baumeister et al., 2015) formalizing both the syntax and the semantics of the two languages and implementing OntoUML models with OntoObject-Z specifications through the mapping of one institution into the other.

ACKNOWLEDGEMENTS

The author thanks four anonymous reviewers for their valuable comments that helped improve the final version of this article.

REFERENCES

- Assar, S. (2014). Model Driven Requirements Engineering: Mapping the Field and Beyond. In *4th International Model-Driven Requirements Engineering Workshop (MoDRE)*, Karlskrona, Sweden. IEEE.
- Aßmann, U., Zschaler, S., and Wagner, G. (2006). *Ontologies, Meta-models, and the Model-Driven Paradigm*. Springer Berlin Heidelberg, Berlin, Heidelberg.
- Baumeister, H., Bettaz, M., Maouche, M., and Mosteghanemi, M. (2015). An Institution for Object-Z with Inheritance and Polymorphism. In *LNCS 8950 (2015)*. Springer.
- Bettaz, M. (2024). Towards an OntoObject-Z Based Ontology for Virtual Local Area Networks, Keynote Speech - Amity Institute of Information Technology.
- Bettaz, M. and Maouche, M. (2017). UML/OCL or Object-Z? In *2017 International Conference on Infocom Technologies and Unmanned Systems (ICTUS'2017)*. IEEE.
- Bettaz, M. and Maouche, M. (2023a). Towards a New Ontology-Based Descriptive Language: OntoObject-Z. In *2023 6th International Conference on Contemporary Computing and Informatics (IC3I)*. IEEE.
- Bettaz, M. and Maouche, M. (2023b). Towards an Ontology for Network Management: Analysis and Refinement. In *2023 10th IEEE International Conference on Electrical, Electronics and Computer Engineering (UPCON)*. IEEE.
- Comer, D. (2015). *Computer Networks and Internets*. Pearson, Edinburgh Gate, 6th edition.
- Diaconescu, R. (2023). Preservation in Many-valued Truth Institutions. *Fuzzy Sets and Systems*, 456:38–71.
- Guizzardi, G. (2005). *Ontological Foundations for Structural Conceptual Models*. Phd thesis, University of Twente, The Netherlands.
- Guizzardi, G., Benevides, A., Fonseca, C., Porello, D., Paulo, J., Almeida, A., and Sales, T. (2021). UFO: Unified Foundational Ontology. In *Applied Ontology 1 (2021)*. IOS PRESS.
- Kim, S. and Carrington, D. (2000). A Formal Mapping Between UML Models and Object-Z Specifications. In *LNCS 1878 (2000)*. Springer-Verlag.
- Odom, W. (2016). *CCENT/CCNA ICND1 100-105 Official Certification Guide, Academic Edition*. Cisco Press, Indianapolis, 2nd edition.
- Pergl, R. (2023). Lectures on Conceptual Modelling, Faculty of Information Technology - Czech Technical University in Prague.
- Roggenbach, M., Cerone, A., Schlingloff, B.-H., Schneider, G., and Shaikh, S. A. (2022). *Formal Methods for Software Engineering*. Springer Cham.
- Seifert, R. and Edwards, J. (2008). *The All-New Switch Book*. Wiley, Indianapolis, 2nd edition.
- Smith, G. (2000). *The Object-Z Specification Language*. Springer.