2017 IEEE 25th International Requirements Engineering Conference Workshops

Towards Using Ontologies for Domain Modeling within the SysML/KAOS Approach

Steve Tueno
and Régine Laleau
LACL
Université Paris-Est Créteil
94010, CRÉTEIL, France
Email: steve.tuenofotso@univ-paris-est.fr
and laleau@u-pec.fr

Amel Mammar
SAMOVAR-CNRS
Télécom SudParis
91000, Evry, France
Email: amel.mammar@telecom-sudparis.eu

Marc Frappier
GRIL
Université de Sherbrooke
Sherbrooke, QC J1K 2R1
Quebec, Canada
Email: Marc.Frappier@usherbrooke.ca

***Abstract*—Modeling the domain of a system to be implemented is a very critical and often neglected activity during requirements engineering. In this paper, we set the scene for an approach to complement the *SysML/KAOS* goal model of a system by adding an ontological representation of its domain knowledge. We think that an Event-B formalization of that domain representation can be used to enrich the formal specifications obtained from the goal model. This paper describes the metamodel that we propose for the representation of domain knowledge and illustrates the proposal through a *Landing Gear System* case study.**

Index Terms*—Requirements Engineering, *SysML/KAOS*, Domain Modeling, Ontologies, *Event-B

I. Introduction

The elicitation, analysis, specification and validation of requirements form the basis of any process aimed at setting up a new computer system. The consequences associated with the occurrence of a failure during any of these steps can be very disastrous and increase exponentially as the development proceeds [1]. The *ANR FORMOSE* project [2] proposes, in view of this observation, to set up a method and a set of tools for the formal modeling of requirements relating to critical complex systems. Formal specifications obtained can thus be formally checked in order to validate the quality of requirements expressed and their consistency with respect to the domain of the system. Prior to this project, a study conducted in [3] investigated the modeling of requirements in the form of goals and proposes the *SysML/KAOS* method. Furthermore, the study reported in [4] has analysed the formalization in *Event-B* [5] of requirements modeled using the *SysML/KAOS* approach. It is clear that only the skeleton of the dynamic part of an *Event-B* specification can be generated from SysML/KAOS goal diagrams. The present work proposes to complement the previous studies by modeling the domain of a system using ontologies. We think that *Event-B* specifications obtained from those ontological models can constitute the data section of the formalization of the *SysML/KAOS* goal model. This paper describes the metamodel that we propose for the representation of domain knowledge and illustrates our proposition through a *Landing Gear System* case study inspired by the one proposed by the *ABZ'14 conference* [6]. The mapping between our domain model representation and *Event-B* is out of the scope of this paper.

The remainder of this paper is structured as follows: Section 2 consists of a synthetic presentation of the SysML/KAOS method and of the Event-B formal language. Follows a presentation, in Section 3, of the relevant state of the art on domain modeling in requirements engineering. In Section 4, we introduce our approach to model the domain of a system specified using the SysML/KAOS method. Finally, Section 5 reports our conclusions and discusses our future work.

II. Background

In this section, we provide a brief overview of *SysML/KAOS*, the *Event-B* formal method and the formalization of *SysML/KAOS* goal diagrams in *Event-B*.

A. *SysML/KAOS*

Requirements engineering focuses on defining and handling requirements. These and all related activities, in order to be carried out, require the choice of an adequate means for requirements representation. The *KAOS* method [1], whose reasons for choosing are explained in [7], proposes to represent the requirements in the form of goals. *KAOS* goals can be *functional* or *non-functional* and are described through five sub-models of which the two main ones are **the object model** which uses the *UML* class diagram for the representation of domain vocabulary and **the goal model** for the determination of requirements to be satisfied by the system and of expectations with regard to the environment through a hierarchy of goals having strategic goals formulated by stakeholders at the root level. The hierarchy is built through a succession of refinements using two operators, ***AND*** and ***OR***. An ***AND refinement*** decomposes a goal into subgoals, and all of them must be achieved to realise the parent goal. Dually, an ***OR refinement*** decomposes a goal into subgoals such that the achievement of only one of them is sufficient for the accomplishment of the parent goal.

KAOS proposes a structured approach to obtaining the requirements based on expectations formulated by stakeholders. Unfortunately, it offers no mechanism to maintain a strong traceability between those requirements and deliverables associated with design and implementation of the system, making it difficult to validate them against the needs formulated. *SysML/KAOS* [3] comes in response to this solicitation by adding to *KAOS* the *SysML UML profile* specially designed by the Object Management Group for the analysis and specification of complex systems. *SysML* allows the capturing of requirements, certainly in a non-formal and ambiguous manner, and the maintaining of traceability links between those requirements and design diagrams resulting from the system design phase. *SysML/KAOS* therefore proposes to extend the *SysML* metamodel with a set of concepts allowing to represent requirements in *SysML* models as *KAOS* goals.

Figure 1 is an excerpt from the landing gear system goal diagram focused on the purpose of landing gear expansion.

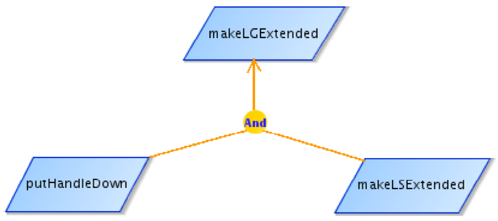

Fig. 1. Excerpt from the landing gear system goal diagram

To achieve the root goal, which is the extension of the landing gear (**makeLGExtended**), the handle must be put down (**putHandleDown**) and landing gear sets must be extended (**makeLSExtended**).

B. From SysML/KAOS to Event-B

Event-B is a formal method defined by *J. R. Abrial* in 2010 for *system modeling* [5]. It is used to prove the preservation of safety invariants about a system. *Event-B* is mostly used for the modeling of closed systems: the modeling of the system is accompanied by that of its environment and of all interactions likely to occur between them. We illustrate our formal model using *B System*, a variant of *Event-B* proposed by *ClearSy*, an industrial partner in the *FORMOSE* project, in its Integrated Development Environment *Atelier B* [8].

The matching between *SysML/KAOS* modeling and *Event-B* specifications is the focus of the work done by [4]. Each layer of abstraction of the goal diagram gives rise to an *Event-B* machine, each goal of the layer giving rise to an event. The refinement links are materialized within the *Event-B* specification through a set of proof obligations and refinement links between machines and between events. Figure 2 represents the *B System* specifications associated with the most abstract layer of the *SysML/KAOS* goal diagram of the Landing Gear System illustrated through Figure 1.

As we can see, the state of the system and the body of events must be manually completed. The state of a system is composed of variables, constrained by an invariant, and constants, constrained by properties. The objective of our study

```
SYSTEM
    LandingGearSystem
EVENTS
    makeLGExtended=
    BEGIN /* extension of the landing gear */
    END
END
```

Fig. 2. Formalization of the root level of the Landing Gear System goal model

is to automatically derive this state in the *Event-B* model using a rigorous modeling of the domain of the system. In the rest of this paper, we focus on domain modeling.

III. Current State on Domain Modeling in Requirements Engineering

A. Presentation

A domain model is a conceptual model capturing the topics related to a specific problem domain [9]. Modeling the domain of a system consists in giving an abstract representation of the set of concepts that the system will be called upon to manipulate and the set of properties and constraints associated with them.

B. Existing Approaches

In [1], the domain of a system is specified by an *object model* described by *UML* class diagrams. An object within this model can be an ***entity*** if it exists independently of the others and does not influence the state of any other object, an ***association*** if it links other objects on which it depends, an ***agent*** if it actively influences the system state by acting on other objects or an ***event*** if its existence is instantaneous, appearing to impulse an update of the state of the system. This approach, which is essentially graphic and little formal as argued in [10] is difficult to exploit in case of critical systems [11]. Moreover, it does not offer mechanisms for referencing a model within another, which limits the reusability of models.

In [7], in addition to *UML* class diagrams, the representation of the system domain involves *UML* object diagrams and ontologies. The case study presented reveals the use of ontologies for the representation of domain knowledge; the model obtained is named the ***domain model***. Furthermore, object and class diagrams are used in the modeling of system structure and constraints and lead to the ***structural model*** which must conform to the *domain model*. This approach, like the previous one, uses *UML* diagrams which are graphical representations, not very formal [10] and slightly expressive [11]. Moreover, the use of several languages for system domain modeling obliges the user to master and manipulate them all, which is an extra source of complexity.

In [11], ontologies are used not only to represent domain knowledge, but also to model and analyze requirements. The proposed methodology is called *knowledge-based requirements engineering (KBRE)* and is mainly used for detection and processing of inconsistencies, conflicts and redundancies among requirements. In spite of the fact that *KBRE* proposes to model domain knowledge through ontologies, the proposal focuses on the representation of requirements and proposes nothing regarding domain modeling.

IV. Our Approach

A. Presentation

An ontology can be defined as a formal model representing concepts that can be grouped into categories through generalization/specialization relations, their instances, constraints and properties as well as relations existing between them. We have chosen to represent domain knowledge using ontologies since they are semantically richer and therefore allow a more explicit and specific representation of domain characteristics.

The vast majority of ontologies are constructed through ***OWL (Ontology Web Language) formalism*** [12] or through ***PLIB (Part LIBrary) formalism*** [13]. *Unfortunately, these formalisms emphasize more on modeling static domain knowledge* [14]. None of these formalisms allows to specify that a knowledge described must remain unchanged or that it is likely to be updated as for example the fact that the state of a landing gear can go from *extended* to *retracted*.

In the following, we propose a metamodel, based on that of *OWL* and *PLIB* for the representation of the domain of a system whose requirements are captured using the *SysML/KAOS* method. Furthermore, this metamodel is designed to allow the specification of knowledge that is likely to evolve over time. It may be supplemented to cover specific use cases.

We present, through Figure 3, the metamodel associated with our domain modeling approach, knowing that yellow elements are those having an equivalent in *OWL* metamodel and that red ones are those that we have either inserted or customized. Furthermore, some constraints and associations, such as the *parentConcept* association, have been extracted from the *PLIB* metamodel. Due to the paper length limitation, we will not highlight all the elements and constraints of the metamodel. The central notion is the notion of *Concept* which represents a group of individuals sharing common characteristics. A *concept* can be declared **variable** (*isVariable=true*) when the set of its individuals is likely to be updated through addition or deletion of individuals. Otherwise, it is considered **constant** (*isVariable=false*). A concept may be associated with another, known as its parent concept, through the *parentConcept* association, from which it inherits properties. As a result, any individual of the child concept is also an individual of the parent concept. The notion of *Relation* is used to capture links between concepts and the notion of *Attribute* links between concepts and data sets (*DataSet*), knowing that data sets are used to group data values (*DataValue*) having the same type. Five sets of data are supported by default: *INTEGER*, *NATURAL* for positive integers, *FLOAT*, *STRING* and *BOOL* for booleans. A relation or an attribute can also be declared *variable* if the list of maplets related to it is likely to change over time. Otherwise, it is considered to be *constant*. The association between a relation and a concept is characterized by the *cardinality* which represents the number of individuals of this concept intervening in a relation instance, *DomainCardinality* for the domain of the relation and *RangeCardinality* for the range. It is bounded by integer constants *minCardinality* which represents the lower bound and *maxCardinality* which represents the upper bound.

Optional characteristics can be specified for a relation : *transitive* (*isTransitive*, default *false*), *symmetrical* (*isSymmetric*, default *false*), *asymmetrical* (*isASymmetric*, default *false*), *reflexive* (*isReflexive*, default *false*) or *irreflexive* (*isIrreflexive*, default *false*). A relation is said to be *transitive* (*isTransitive=true*) when the relation of an individual x with an individual y which is in turn in relation to z results in the relation of x and z. It is said to be *symmetric* when the relation between an individual x and an individual y results in the relation of y to x. It is said to be *asymmetric* when the relation of an individual x with an individual y has the consequence of preventing a possible relation between y and x. It is said to be *reflexive* when every individual of the domain is in relation with himself. It is finally said to be *irreflexive* when it does not authorize any connection of an individual of the domain with himself. Furthermore, an attribute can be *functional* (*isFunctional*, default *true*) if it associates to each individual of its domain one and only one data value of its range.

The notion of *Rule* is used to represent constraints between different elements of the domain model in the form of *Horn clauses*: each rule has a body which represents its antecedent and a head which represents its consequent, body and head designating conjunctions of atoms.
Each domain model is associated with a level of refinement of the *SysML/KAOS* goal diagram and is likely to have as its parent, through the *parent* association, another domain model. It should be noted that the parent domain model must be associated with the refinement level of the *SysML/KAOS* goal diagram directly above the refinement level to which the child domain model is associated.

In order to be able to be used in the setting up of large complex systems, *SysML/KAOS* allows the refinement of a leaf of a goal diagram in another diagram having this goal as root. For example, in Figure 4, the goal *G3*, which is a leaf of the first goal diagram, is the root of the second one. When this happens, we associate to the most abstract level of the new goal diagram the domain model associated with the most concrete level of the previous goal diagram as represented in Figure 4: *Domain Model 2*, which is the domain model associated to the most concrete level of the first diagram, is also the domain model associated to the root of the second one.

B. Illustration

Our *Landing Gear System* case study deals with the landing gear system of an aircraft which is retracted (respectively extended) through a handle which is manually put up (respectively down). We assume that each aircraft has one landing gear system which is equipped with three landing sets which can be extended or retracted. We also assume that at the initial state, there is one landing gear named *LG1* which is extended

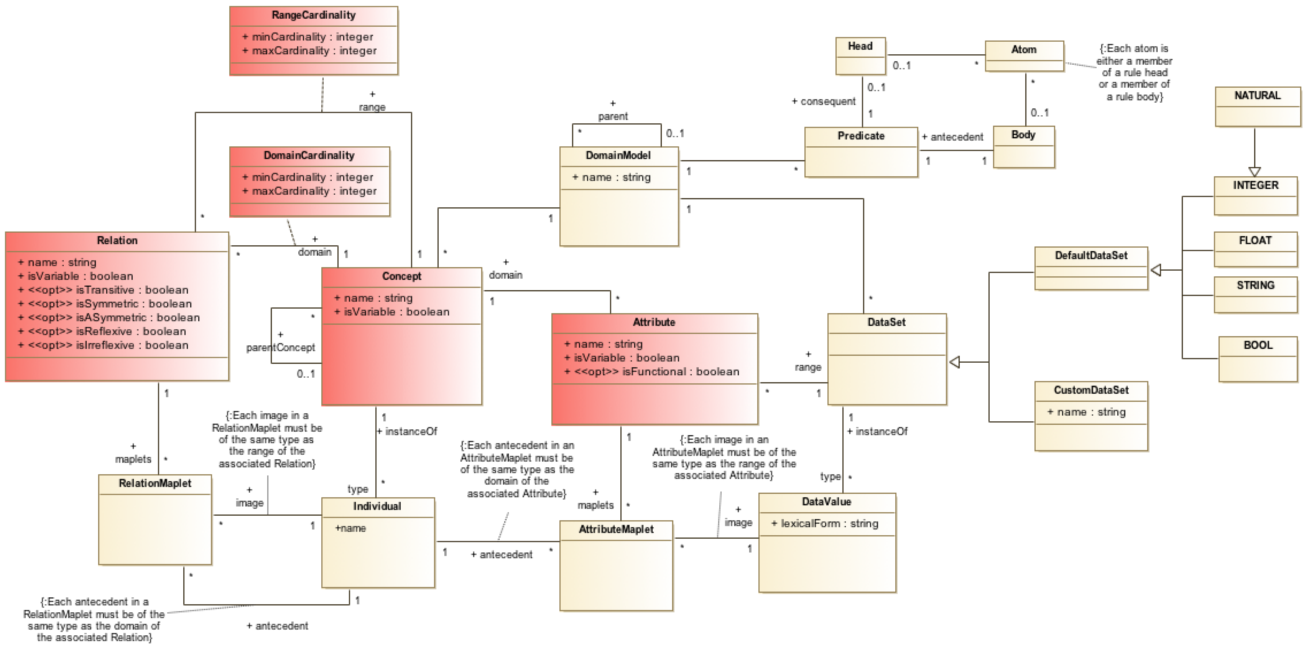

Fig. 3. Our metamodel associated with domain modeling

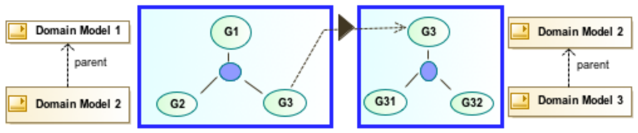

Fig. 4. Management of the partitioning of a *SysML/KAOS* goal model

and is associated to one handle named *HD1* which is down and to landing sets *LS1*, *LS2* and *LS3* which are all extended.

We have identified two graphical syntaxes for the representation of ontologies : the syntax proposed by *OntoGraph* [15] and the syntax proposed by *OWLGred* [16]. The *OntoGraph* syntax is the one used in [7]. Unfortunately, it does not allow the representation of some domain model elements such as attributes or cardinalities. For our case study, we have thus decided to use the *OWLGred* syntax. Furthermore, for readability purposes, we have decided to remove optional characteristics representation such as *isTransitive* or *isSymmetric*.

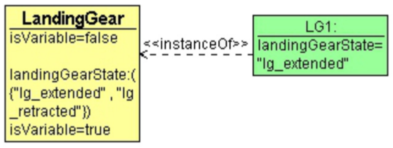
LandingGear
isVariable=false
landingGearState:({"lg_extended" , "lg_retracted"})
isVariable=true
<<instanceOf>>
LG1:
landingGearState="lg_extended"

Fig. 5. ***untitled-ontology-44***: ontology associated to the root level

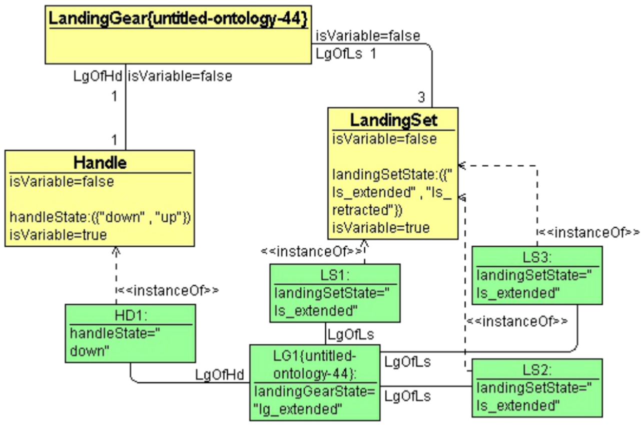
LandingGear{untitled-ontology-44}
isVariable=false
LgOfLs 1
LgOfHd
isVariable=false
1
1
3
LandingSet
isVariable=false
landingSetState:({"ls_extended" , "ls_retracted"})
isVariable=true
Handle
isVariable=false
handleState:({"down" , "up"})
isVariable=true
<<instanceOf>>
<<instanceOf>>
<<instanceOf>>
<<instanceOf>>
LS1:
landingSetState="ls_extended"
LS3:
landingSetState="ls_extended"
HD1:
handleState="down"
LgOfLs
LG1{untitled-ontology-44}:
landingGearState="lg_extended"
LgOfHd
LgOfLs
LgOfLs
LS2:
landingSetState="ls_extended"

Fig. 6. ***untitled-ontology-45***: ontology associated to the first level of refinement

Figures 5 and 6 represent respectively the domain model associated to the root level of the landing gear system goal model, illustrated through Figure 1, and that associated with the first refinement level. The domain model associated to the goal diagram root level is named *"untitled-ontology-44"* and the one associated to the first refinement level is named *"untitled-ontology-45"*.

In *untitled-ontology-44*, The *landing gear entity* is modeled as an instance of *Concept* named *"LandingGear"*. Since it is impossible to dynamically add or remove a landing gear, the attribute *isVariable* of ***LandingGear*** is set to *false*. The possible states of a landing gear is modeled as an instance of *Attribute* named *"landingGearState"*, having ***LandingGear*** as domain and as range an instance of *DataSet* containing two instances of *DataValue* of type *STRING*: ***"lg_extended"*** for the extended state and ***"lg_retracted"*** for the retracted state. *"landingGearState"* is graphically represented in ***LandingGear*** and, since it is possible to dynamically change a landing gear state, its *isVariable* attribute is set to *true*. *LG1* is modeled as an instance of *Individual* named *"LG1" «instanceOf»* ***LandingGear*** and to which the attribute ***landingGearState*** associates the instance of *DataValue* named *"lg_extended"*, since *LG1* is extended at the initial state.

As can be seen, domain model ontology associated with a level of refinement of the *SysML/KAOS* goal model imports the ontology of its parent domain model associated with the previous level in order to access all domain elements modeled in it: in *untitled-ontology-45* represented in Figure 6, elements defined in *untitled-ontology-44* represented in Figure 5 are imported and reused. The association of landing sets to landing gear is modeled as an instance of *Relation* named *"LgOfLs"*. Since the association of a landing set to a landing gear cannot be changed dynamically, the attribute *isVariable* of ***LgOfLs*** is set to *false*. The instance of *DomainCardinality* associated to ***LgOfLs*** has its *minCardinality* attribute and its *maxCardinality* attribute set to *1*. On the other side, the instance of *RangeCardinality* associated to ***LgOfLs*** has its *minCardinality* attribute and its *maxCardinality* attribute set to *3*. There are three instances of *RelationMaplet* to model the association of *LG1* to *LS1*, *LS2* and *LS3*, each having as *image* ***LG1*** and as *antecedent* the corresponding ***LandingSet*** instance.

V. Conclusion and Future Works

In order to allow the automatic generation of the system state in the *Event-B* specification of a *SysML/KAOS* goal model, we are interested in modeling the domain of systems. A detailed study of domain modeling approaches led us to choose ontologies as a means of representing the system domain.

The paper allowed us to present the context underlying our study and to draw up a rapid state of the art related to our problem. After positioning ourselves as to the existing, we have presented the metamodel that we propose for the specification of domain model ontologies and we have exemplified our approach through a ***Landing Gear System*** case study.

Work in progress is aimed at developing mechanisms to automatically complete the *Event-B* model resulting from the formalization of the *SysML/KAOS* goal diagram using domain knowledge modeled as ontologies following our proposal and at integrating our approach within the open-source platform *Openflexo* [17]. We are also interested in an approach allowing to automatically validate the consistency of an ontology with respect to its associated goals expressed in the *SysML/KAOS* goal model.

Acknowledgment

This work is carried out within the framework of the *FORMOSE* project [2] funded by the French National Research Agency (ANR).

References

[1] A. van Lamsweerde, *Requirements Engineering - From System Goals to UML Models to Software Specifications*. Wiley, 2009.

[2] ANR-14-CE28-0009, “Formose ANR project,” 2017. [Online]. Available: http://formose.lacl.fr/

[3] C. Gnaho and F. Semmak, “Une extension SysML pour l’ingénierie des exigences dirigée par les buts,” in *Actes du XXVIIIème Congrès INFORSID, Marseille, France, 25-28 mai 2010*, 2010, pp. 277–292.

[4] A. Matoussi, F. Gervais, and R. Laleau, “A goal-based approach to guide the design of an abstract Event-B specification,” in *16th IEEE International Conference on Engineering of Complex Computer Systems, ICECCS 2011, Las Vegas, Nevada, USA, 27-29 April 2011*, I. Perseil, K. K. Breitman, and R. Sterritt, Eds. IEEE Computer Society, 2011, pp. 139–148. [Online]. Available: http://dx.doi.org/10.1109/ICECCS.2011.21

[5] J. Abrial, *Modeling in Event-B - System and Software Engineering*. Cambridge University Press, 2010. [Online]. Available: http://www.cambridge.org/uk/catalogue/catalogue.asp?isbn=9780521895569

[6] Y. A. Ameur and K. Schewe, Eds., *Abstract State Machines, Alloy, B, TLA, VDM, and Z - 4th International Conference, ABZ 2014, Toulouse, France, June 2-6, 2014. Proceedings*, ser. Lecture Notes in Computer Science, vol. 8477. Springer, 2014. [Online]. Available: http://dx.doi.org/10.1007/978-3-662-43652-3

[7] A. Mammar and R. Laleau, “On the use of domain and system knowledge modeling in goal-based Event-B specifications,” in *Leveraging Applications of Formal Methods, Verification and Validation: Foundational Techniques - 7th International Symposium, ISoLA 2016, Imperial, Corfu, Greece, October 10-14, 2016, Proceedings, Part I*, ser. Lecture Notes in Computer Science, T. Margaria and B. Steffen, Eds., vol. 9952, 2016, pp. 325–339. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-47166-2_23

[8] ClearSy, “Atelier B: B System,” 2014. [Online]. Available: http://tools.clearsy.com/tutorials/atelier-b-4-x-in-a-nutshell/

[9] M. Broy, *Domain Modeling and Domain Engineering: Key Tasks in Requirements Engineering*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 15–30. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-37395-4_2

[10] W. E. McUmber and B. H. C. Cheng, “A general framework for formalizing UML with formal languages,” in *Proceedings of the 23rd International Conference on Software Engineering, ICSE 2001, 12-19 May 2001, Toronto, Ontario, Canada*, H. A. Müller, M. J. Harrold, and W. Schäfer, Eds. IEEE Computer Society, 2001, pp. 433–442. [Online]. Available: http://dx.doi.org/10.1109/ICSE.2001.919116

[11] T. H. Nguyen, B. Q. Vo, M. Lumpe, and J. Grundy, “KBRE: a framework for knowledge-based requirements engineering,” *Software Quality Journal*, vol. 22, no. 1, pp. 87–119, 2014. [Online]. Available: http://dx.doi.org/10.1007/s11219-013-9202-6

[12] K. Sengupta and P. Hitzler, “Web ontology language (OWL),” in *Encyclopedia of Social Network Analysis and Mining*, 2014, pp. 2374–2378. [Online]. Available: http://dx.doi.org/10.1007/978-1-4614-6170-8_113

[13] G. Pierra, “The PLIB ontology-based approach to data integration,” in *Building the Information Society, IFIP 18th World Computer Congress, Topical Sessions, 22-27 August 2004, Toulouse, France*, ser. IFIP, R. Jacquart, Ed., vol. 156. Kluwer/Springer, 2004, pp. 13–18. [Online]. Available: http://dx.doi.org/10.1007/978-1-4020-8157-6_2

[14] L. Zong-yong, W. Zhi-xu, Z. Ai-hui, and X. Yong, “The domain ontology and domain rules based requirements model checking,” *International Journal of Software Engineering and Its Applications*, vol. 1, no. 1, pp. 89–100, 2007.

[15] S. Falconer, “Protégé - ontograph,” 2010. [Online]. Available: http://protegewiki.stanford.edu/wiki/OntoGraf

[16] I. UL, “Owlgred home,” 2017. [Online]. Available: http://owlgred.lumii.lv/

[17] Openflexo, “Openflexo project,” 2015. [Online]. Available: http://www.openflexo.org