

# Modeling Object Oriented Systems via Controlled English Verbalization of Description Logic

Pawel Kaplanski

Gdansk University of Technology

{[pawel.kaplanski@gmail.com](mailto:pawel.kaplanski@gmail.com)}

**Abstract.** The need for formal methods for Object Oriented (OO) systems resulted in methods like UML and Lepus3 that are de-facto graphical languages equipped with formal tools that are able to handle the design of OO systems. However, they lack precise semantics which might lead to problems, such as inconsistencies or redundancies. On the other hand, to our knowledge, there is no approach that allows one to understand and follow the requirements of a design-implementation path for people that lack knowledge about standard software modeling language. The approach to OO system modeling presented in this paper uses Controlled English (CE) (a well defined subset of English) in the area where graphical languages are currently used. Object Oriented Constructs are modeled first in Description Logic (DL) that provides the logical framework and the CE verbalization of DL (CE<sub>DL</sub>) finally bridges DL with CL allowing to access OO world in formal, yet understandable way for both human and computer.

## 1 Introduction

Rapid development of software engineering and software production methodologies, which took place as complexity of information systems advanced, is connected with the need for formal description methods for knowledge acquired during their development [1]. SEMAT [[www.semat.org](http://www.semat.org)] initiative identifies “Methods&Tools” as one of macro-trends in modern Software Engineering. That macro-trend resulted already in the unified modeling language UML/OCL [14][15] which is a standard software modeling language nowadays. LePUS3[7]—another formal specification language designed to capture and convey the building-blocks of object-oriented design—is also tailored to integrate the strength of specification and modeling notations. It also proved its expectations, e.g. in automatic verification of Design Patterns [6].

The approach to OO system modeling presented in this paper differs from the above approaches as it uses the Controlled English<sup>1</sup> [5][10] (CE) as a software modeling language. The motivation lies in the pragmatic observation of industrial

---

<sup>1</sup> A subset of natural English, obtained by restricting the grammar and vocabulary in order to reduce or eliminate ambiguity and complexity

need for a “human-readable language” that emerged together with the increasing complexity of computer programs. To understand the software structure one is required to have a background in the field of a computer science, especially in software modeling. It is hard to trace a software structure and measure it for “non software oriented” personal-authority that is forced to use a graphical software-modeling language, without prior education in the field. What is more, without the support of formal-methods it is almost impossible to trace and understand the consequences of even small changes of design in a complex software system. In consequence, strategic decisions that are made by the authorities reveal a lack of information about the real state of the software product that is developed within the organization. Moving further, one can consider modern software-intensive systems as made of three kinds of participants: software, hardware and bioware<sup>2</sup>. While communication between software and hardware is realized by the computer-code, a programming language bridges software components with bioware. It is obvious that the natural language would allow larger community to access the software. In this paper such an approach is proposed.

Software structures (especially OO) can be treated as ontologies<sup>3</sup>. Description Logic (DL) is a subset of first order logic focused on ontology formalization and has the very important property of being computable. Computability ensures that reasoning tasks<sup>4</sup> of DL can be made in finite time and space. Some dialects of DL (e.g. EL++ [2]) give a promise that this task can be done in polynomial time, others—more expressive, (e.g. SROIQ [9] )—use optimization techniques for most common cases. The ontological framework, to be useful, needs to be responsive<sup>5</sup> and therefore the selection for the formalism is a curtail requirement for software modeling tasks. Recently it was discovered that even if the DL has a different semantic than OO modeling languages have<sup>6</sup> it is still possible to emulate curtail parts of OO within DL [11][3]. In this paper first, the correspondence to software structure is done via SROIQ DL (because of its expressive power) and finally through the CE verbalization of DL ( $CE_{DL}$ ) these two semantic technologies merge together.

## 2 Verbalization of DL

Although DL is now most often associated with the semantic-web, new applications appeared recently proving the usability of DL also in other fields of

---

<sup>2</sup> Bioware is a neologism for the human that interacts with the software-intensive system.

<sup>3</sup> The source-code of a computer program is a set of sentences describing what to do with data and therefore they form an ontology of program behavior and data.

<sup>4</sup> Reasoning tasks include for example.: concept classification, that is, a hierarchical arrangement of concepts within the notion of includes. Another one is classification of instances to certain concepts.

<sup>5</sup> Responsiveness – the ability of a computer system to perform an assigned function within the required time interval

<sup>6</sup> DL is equipped with open-world assumption and it lacks defaults while OO uses closed-world assumption and uses defaults to describe a class-inheritance.

interest. These applications include DL verbalizations in CE ( $CE_{DL}$ ) that enable access to DL in natural language as a part of human-computer interface. Very expressive CE like ACE [5] can also be used to verbalize DL. ACE as a very powerful CE that can be translated into a non-decidable subset of first-order logic provides also its subset called  $ACE_{OWL}$ [10] that can be translated into OWL 2 (equivalent of  $SROIQ(D)$  DL). On the other hand most of OWL 2 can be translated into a subset  $ACE_{OWL}$ . It was recently shown [12] that  $ACE_{OWL}$  is more natural for people than formal-looking  $CE_{DL}$  syntaxes (like Manchester [8]/Sydney [4] OWL Syntax).

### 3 Modeling OO with $CE_{DL}$

The research for  $CE_{DL}$  described in this paper was inspired by  $ACE_{OWL}$ , however the grammar of CE used in this research (called  $LL_{(1)}CE_{DL}$ ), was implemented using  $LL(1)$  top-down parser generator [13] and equipped with additional features that are not implemented within ACE<sup>7</sup>. Therefore the sentences of  $LL_{(1)}CE_{DL}$  might not be a valid expressions in ACE (and even in English) because of limitations of a context-free  $LL(1)$  grammar, nevertheless the BNF rules of the  $LL_{(1)}CE_{DL}$  grammar were designed to be as close as possible to  $ACE_{OWL}$ .

Object (the main idea used in OO languages) and class (specification for object construction) together with carefully selected relations (like extend or materialize) are modeled here first in DL and then the adequate  $LL_{(1)}CE_{DL}$  sentence is created. The existence of a bidirectional  $LL_{(1)}CE_{DL} \leftrightarrow DL$  mapper, that is a part of a parser, bridges these two technologies together.

If for each class, object, method and attribute a corresponding instance of concept is assigned then the basic model of the object can be created in DL (see Fig.1). Next, the relations between classes can be established. Classes can extend<sup>8</sup> each other to provide a subtyping mechanism<sup>9</sup>. On the other hand objects realize every type related to their class and therefore adequate properties on selected relations are required. Using selected relationships it is possible to specify requirements for a class. E.g.: to specify that a class is abstract<sup>10</sup>, to require that the class must be a root of a class hierarchy or to prevent the class from being inherited<sup>11</sup>. One can also require that a class is an implementation of a singleton design pattern, explicitly requiring that it can have one and only one unique realization. It is also possible to handle sets of classes e.g.: one can compute the set of classes that form a branch of the class hierarchy and

---

<sup>7</sup> Additional features include: “A is equivalent to B” construction that correspond to  $A \equiv B$  DL expression, “the C” that corresponds to unnamed individual of concept C and allow use of parentheses that allows production of more complex expressions in CE.

<sup>8</sup> I use the term “extend” in place of “subclass” because it has more intuitive meaning about the source and the target than a subclass relationship (“B subclasses A” is not so intuitive as “B extends A”) and is easier to understand for a non-expert.

<sup>9</sup> Subtyping mechanism (partial ordering over types) is a core idea behind the paradigm of a OO abstraction.

<sup>10</sup> A class that cannot have any instances.

<sup>11</sup> Such classes are called “final” in Java.

hierarchy itself. Having the ability to express relations between class hierarchies, the opportunity to describe structural relationships of design-pattern is opening. It is even possible to set up the architectural design-constraints like layer-separation that then is able to be computed by DL Reasoner.

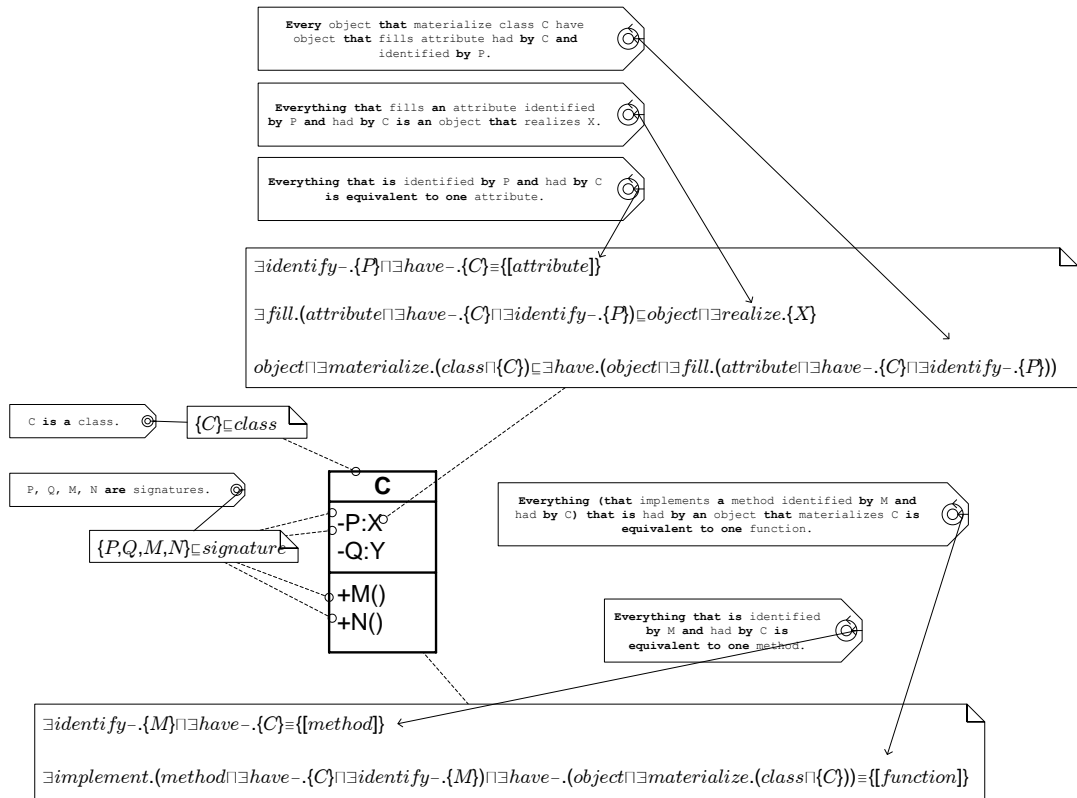


Fig. 1. The class and its DL model

#### 4 Conclusion and Future Work

This paper presents the results of research on the potential application of Description Logic verbalized by Controlled English inspired by ACE<sub>OWL</sub> and implemented with LL(1) grammar (called LL<sub>(1)</sub>CE<sub>DL</sub>) as a modeling language for Object Oriented (OO) systems. It was shown that LL<sub>(1)</sub>CE<sub>DL</sub> can be used to describe basic OO constructs. The research made so far gives hope that CE in general enables us to use one and the same system for storing requirements, as well as the project and system architecture, which in turn ensures the logical cohesion of artifacts created in different stages of software development. Moreover, in such a case, it would be possible to maintain a coherent terminological base between different groups of people involved in the information project through a common knowledge

management system. These possibilities are currently being intensively investigated by the author of this article. To compare the usage of CE as an OO modeling language in comparison to existing methodologies that use the graphical languages (UML/LePUS3) the evaluation is also planned.

## References

- [1] Harold Abelson and Gerald J. Sussman. *Structure and Interpretation of Computer Programs*. MIT Press, Cambridge, MA, USA, 1996.
- [2] F. Baader, C. Lutz, and B. Suntisrivaraporn. Efficient reasoning in  $EL^+$ . In *Proceedings of the 2006 International Workshop on Description Logics (DL2006)*, CEUR-WS, 2006. To appear.
- [3] Daniela Berardi, Diego Calvanese, and Giuseppe De Giacomo. Reasoning on uml class diagrams is exptime-hard. In *Proc. of the 16th Int. Workshop on Description Logic (DL 2003)*, volume 81 of *CEUR Electronic Workshop Proceedings*, <http://ceur-ws.org/>, pages 28–37, 2003.
- [4] Anne Cregan, Rolf Schwitter, and Thomas Meyer. Sydney OWL Syntax - towards a controlled natural language syntax for OWL 1.1. In *OWLED*, 2007.
- [5] Norbert E. Fuchs, Uta Schwertel, and Rolf Schwitter. Attempto Controlled English - not just another logic specification language. In *LOPSTR '98: Proceedings of the 8th International Workshop on Logic Programming Synthesis and Transformation*, pages 1–20, London, UK, 1990. Springer-Verlag.
- [6] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design patterns: elements of reusable object-oriented software*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1995.
- [7] Epameinondas Gasparis, Jonathan Nicholson, and Amnon H. Eden. Lepus3: An object-oriented design description language. In *Diagrams '08: Proceedings of the 5th international conference on Diagrammatic Representation and Inference*, pages 364–367, Berlin, Heidelberg, 2008. Springer-Verlag.
- [8] Matthew Horridge, Nick Drummond, John Goodwin, Alan L. Rector, Robert Stevens, and Hai Wang. The manchester OWL Syntax. In *OWLED*, 2006.
- [9] Ian Horrocks, Oliver Kutz, and Ulrike Sattler. The even more irresistible sroiq. In Patrick Doherty, John Mylopoulos, and Christopher A. Welty, editors, *KR*, pages 57–67. AAAI Press, 2006.
- [10] Kaarel Kaljurand. *Attempto Controlled English as a Semantic Web Language*. PhD thesis, Faculty of Mathematics and Computer Science, University of Tartu, 2007.
- [11] Seiji Koide, Jans Aasman, and Steve Haflich. OWL vs. object oriented programming. In *Workshop on Semantic Web Enabled Software Engineering (SWESE)*, November 2005. Galway, Ireland.
- [12] Tobias Kuhn. How to evaluate controlled natural languages. *CoRR*, abs/0907.1251, 2009.
- [13] D. J. Rosenkrantz and R. E. Stearns. Properties of deterministic top down grammars. In *STOC '69: Proceedings of the first annual ACM symposium on Theory of computing*, pages 165–180, New York, NY, USA, 1969. ACM.
- [14] James Rumbaugh, Ivar Jacobson, and Grady Booch. *The Unified Modeling Language Reference Manual*. Addison-Wesley, Boston, MA, 2. edition, 2005.
- [15] Jos Warmer and Anneke Kleppe. *The Object Constraint Language: Precise Modeling with UML*. Addison-Wesley, Reading, MA, 1999.