Making Domain Knowledge Explicit using $\mathcal{SHIQ}$ in Object-Oriented Software Development

Ragnhild Van Der Straeten
System and Software Engineering Lab (SSEL)
Vrije Universiteit Brussel, Belgium
Email: rvdstrae@vub.ac.be

Abstract

In object-oriented software development, domain knowledge is implicitly present in the different models and in the head of the developers. We propose to use the Description Logic $\mathcal{SHIQ}$ and extensions to make this knowledge explicit and to support the software modeler in using this knowledge.

1 Introduction

To be able to develop a software application, the software developer must have a thorough knowledge of the application domain. A domain is *some area of interest and can be hierarchically structured* [10]. Domain knowledge *specifies domain specific knowledge and information types that we want to talk about in an application* [10]. In this paper we will focus on how implicit domain knowledge can be made explicit and how this explicit knowledge can be used to support the software developer through the software development life cycle (SDLC).

Nowadays, the de-facto modeling language used in object-oriented development is the Unified Modeling Language (UML) [3]. The visual representation of the UML consists of several diagram types. Within the SDLC several levels of abstraction are used. In one abstraction level several kinds of UML diagrams are used, e.g. class diagrams, state diagrams, interaction diagrams. We will call one such layer a model of the application.

A lot of domain knowledge is implicitly and explicitly present in the models used throughout the SDLC and in the different UML diagrams. However this knowledge is not always entirely used. E.g. if the information provided by the class diagram is properly formalized, consistency of the class diagram and class consistency can be checked. The same is true for state diagrams, e.g. it is possible to check if only one transition can be taken out of a given state or to check if the state diagram is deadlock free. In this case, the explicit knowledge can be used to check consistency of a diagram. An example is given in section 3.2.

A lot of domain knowledge is only present in the head of the software developers or is

lost through the SDLC. To make this domain knowledge explicit, the Object Constraint Language (OCL) [4] can be used. OCL constraints can be used e.g. to specify class invariants on class diagrams or to specify guards on state diagrams. However, we also want to reason about these constraints and e.g. check the constraints w.r.t. the information present in one or more diagram types. We will use one language for this and translate the different diagrams and write the additional domain knowledge down in this language. This also allows the semantic linking of the different diagram types. For example, guards on a state diagram can be checked w.r.t. the knowledge present in class diagrams. An example is given in section 3.1.

As language we propose to use the $\mathcal{SHIQ}$ Description Logic. In the next section, this choice is motivated and the translation of class diagrams, state diagrams and OCL constraints is briefly discussed. In section 3 examples are given of the use of explicit domain knowledge. Section 4 concludes this paper.

2 The Use of $\mathcal{SHIQ}$

There are several reasons for choosing the Description Logic (DL) family to make more explicitly use of domain knowledge. Object-oriented programming languages originate from frame-based systems. Description logics are based on the same ideas as semantic networks and frames but provide these with exact semantics. DLs have already proven their use in knowledge representation and reasoning. A DL consists of a description language, a knowledge specification language and automatic reasoning procedures. The most important aspect is these automatic reasoning procedures. They allow to reason about the consistency of knowledge bases. Apart from their classical application in knowledge representation, they are used in various applications such as database applications [7] and as ontology language for the semantic web [8]. In the next subsections, we show the translation of UML class diagrams, state diagrams and constraints written in the Object Constraint Language (OCL) into the DL $\mathcal{SHIQ}$. For the syntax and semantics of this DL we refer to [11]. In [6], class diagrams are translated and in [13], UML state diagrams and OCL expressions are translated into the DL $\mathcal{DLR}$. This DL is less expressive than $\mathcal{SHIQ}$ and can be mapped to $\mathcal{SHIQ}$. So, we will not have a detailed look at these translations, but only consider the mapping of the basic concepts of these diagrams.

2.1 Class Diagrams

- A class is represented by an $\mathcal{SHIQ}$ concept.

- An attribute of a class is translated as a binary relation. To specify the type of the attribute, the value type restriction is used on that binary relation. E.g. an attribute a of type T of a class C can be specified as follows: $C \sqsubseteq \forall a.T$. In UML it is possible to specify the multiplicity of an attribute of a class. This multiplicity $[i, j]$ can be expressed in $\mathcal{SHIQ}$ by number restrictions. The multiplicity of the attribute a can be asserted as follows:

$C \sqsubseteq (\geq i\ a.T) \sqcap (\leq j\ a.T)$.

- An operation of a class consists of an operation name, a parameter list $(P_1, ..., P_n)$ and a return list $(R_1, ..., R_m)$. This can be formalized as follows: $C \sqsubseteq \forall op.(operation \sqcap \forall u_1.P_1 \sqcap ... \forall u_n.P_n \sqcap \forall u_{n+1}.R_1 \sqcap ... \sqcap \forall u_{n+m}.R_m)$.

- UML associations (and aggregations) represent relationships between instances of classes. In most cases, associations are binary relations. These can be translated to $\mathcal{SHIQ}$ roles in a straightforward way. An UML association class can be represented by a concept with two binary relations and the extra constraint that only one instance of the association class can be present between any two participating objects, must also be asserted.

- Generalization is naturally supported in $\mathcal{SHIQ}$. The fact that the class B is a generalization of the class A can be expressed as follows: $A \sqsubseteq B$.

2.2 State Diagrams

A state is mapped onto a primitive concept in $\mathcal{SHIQ}$. A guard is a logical condition and is translated as a logical expression of concepts. An event is also mapped onto a concept. A transition is a binary relation between two states, a start state and an end state. This is represented as a binary relation in $\mathcal{SHIQ}$ between the two involved concepts. Actions are modeled as part of the transition. In section 3.1, an example is given.

2.3 OCL Constraints

We will only have a look at OCL basic types, because state-of-the-art DLs do not support this knowledge. For the translation of the collection types and the navigation paths we refer to [13]. In OCL the basic types are `Integer`, `String`, `Boolean` and `Real`. To be able to represent e.g. numerical knowledge, a DL must be extended with concrete domains [5]. OCL basic types can be represented by concrete datatypes and a mechanism can be provided to derive new datatypes as done in [11]. However, these concrete domains are unary. In [12], $\mathcal{SHIQ}$ is extended with one particular concrete domain, whose domain are the rationals and which has binary predicates $\{\leq, \geq, \neq, =, <, >\}$ and unary predicates P_q for each $q \in Q$ with $P \in \{\leq, \geq, \neq, =, <, >\}$. The following concept: $\texttt{Customer} \sqcap \exists \texttt{age}.\ \geq_{18}$ describes customers whose age is 18 or more. To be able to map OCL constraints to $\mathcal{SHIQ}$ expressions, this logic has to be extended with the appropriate concrete domains.

3 Examples

In this section we will give two examples of how $\mathcal{SHIQ}$ can be used to make knowledge explicit and how this knowledge can be used to support the software developer. In the first example, using the reasoning mechanisms of the DL the consistency is checked between a class diagram and a state diagram. In the second example, implicit knowledge is made explicit which enables a more thorough support for the modeler.

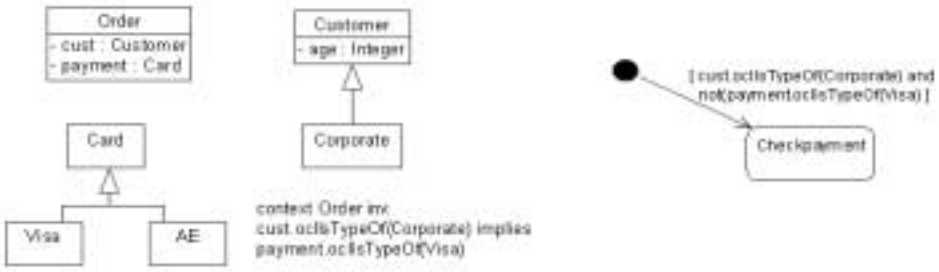

Figure 1: Class Diagram and State Diagram.

3.1 Consistency between Several UML Diagram Types

$\mathcal{SHIQ}$ has been implemented in efficient DL systems such as FaCT [2] and RACER [1]. We will use here the FaCT system. In figure 1 a class diagram is shown consisting of the classes `Order`, `Customer`, `Corporate`, `Card`, `Visa` and `AE`. The `Order` class has two attributes, `cust` and `payment`. The attribute `cust` indicates the kind of `Customer` associated with the `Order`. A `Customer` can be a `Corporate` customer. The attribute `payment` indicates which kind of `Card` is used. A card can be a `Visa` or an `AE` (American Express) card. The OCL class invariant indicates that a corporate customer can only pay with a Visa card. This class invariant must be true at any moment in time. Next to this class diagram, a piece of the state diagram for the `Order` class is shown. To be able to go from the start state to the `Checkpayment` state, the specified guard should be true. However this guard is in contradiction with the OCL class invariant. So, it is not possible to go from the start state to the `Checkpayment` state. This

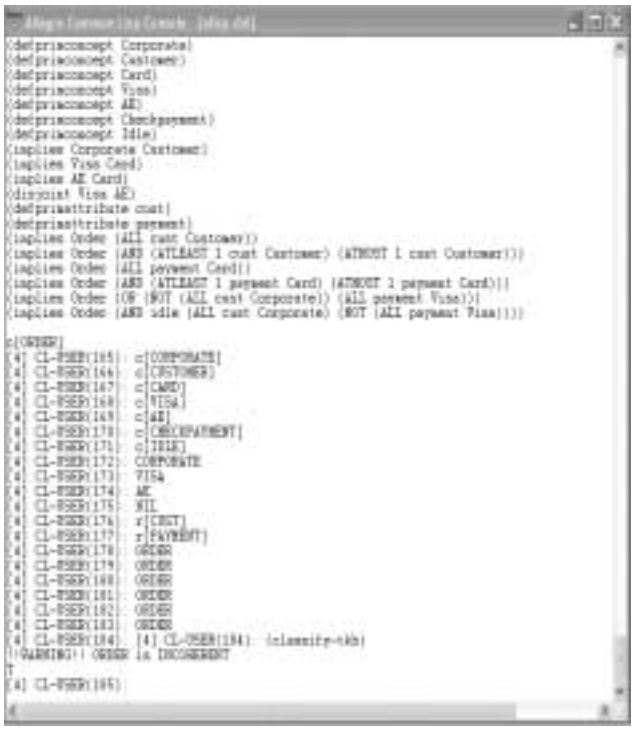

Figure 2: A FaCT session.

can be checked by FaCT. In figure 2 the translation of the class diagram and state diagram is shown. If these definitions are checked on consistency as done in the session in figure 2 by calling the `classify-tkb` procedure, the system reports the inconsistency of the `Order` concept [1].

This lets us conclude that the different translations will enable us to couple the class diagrams, corresponding state diagrams and OCL constraints in a model.

3.2 Making Implicit Knowledge in UML Class Diagrams Explicit

Consider the organization hierarchies in figure 3 [9]. The operating units of the organization are divided in regions and these are divided in Divisions which are divided in sales offices. This structure can be modeled by the first class diagram of figure 3. However, if the organization changes, the model must be changed. Another model which is simpler to change is presented in the second class diagram of figure 3. However by modeling the organization structure in that way, some knowledge is made implicit, nl.: 1. An operating unit cannot have a parent. 2. The parent of a region must be an operating unit. 3. The parent of a division must be a region. 4. The parent of a sales office must be a division. This knowledge is made explicit again by writing it down in $\mathcal{SHIQ}$ together with the translation of the class diagram. This explicit domain knowledge enables the support of the modeling of software applications. For example, while modeling the application, the underlying reasoning mechanism notifies if one of the rules is being violated.

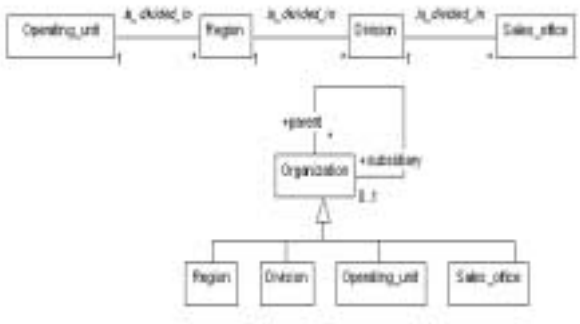

Figure 3: Explicit Knowledge made Implicit and vice versa.

4 Discussion and Future Work

Our goal is to build an intelligent modeling tool which enables to make implicit knowledge explicit and to use this knowledge to support the designer in designing software applications. The formal framework underlying this tool will be based on DL. This provides the developer with the necessary reasoning capabilities enabling the reasoning about different models and phases of the SDLC. This has the advantages that the co-evolution of the different phases of the SDLC is more guaranteed, reuse and adaptability of software is improved and the understandability of the software designs increases.

[1] FaCT does not provide any explanation of its inferences, although this would be useful in software modeling.

References

[1] RACER. Volker Haarslev and Ralf Möller, http://kogs.www.informatik.uni-hamburg.de/~race/.

[2] The FaCT System. Ian Horrocks, http://www.cs.man.ac.uk/~horrocks/FaCT/.

[3] The OMG Unified Modeling Language Specification. The Object Management Group http://www.omg.org.

[4] Kleppe A. and Warmer J. *The Object Constraint Language: Precise Modeling with UML.* Addison Wesley, 1999.

[5] F. Baader and P. Hanschke. A Scheme for Integrating Concrete Domains into Concept Languages. In *Proceedings of the 12th International Joint Conference on Artificial Intelligence, IJCAI-91*, pages 452–457, Sydney (Australia), 1991.

[6] Andrea Calí, Diego Calvanese, Giuseppe De Giacomo, and Maurizio Lenzerini. Reasoning on UML Class Diagrams in Description Logics. In *Proc. of IJCAR Workshop on Precise Modelling and Deduction for Object-oriented Software Development (PMD 2001)*, 2001.

[7] Diego Calvanese, Maurizio Lenzerini, and Daniele Nardi. Description logics for conceptual data modeling. In Jan Chomicki and Gnter Saake, editors, *Logics for Databases and Information Systems*, pages 229–263. Kluwer Academic Publisher, 1998.

[8] D. Fensel, I. Horrocks, F. van Harmelen, S. Decker, M. Erdmann, and M. Klein. OIL in a nutshell. In R. Dieng, editor, *Proceedings of the 12th European Workshop on Knowledge Acquisition, Modeling, and Management (EKAW'00)*, number 1937 in Lecture Notes In Artificial Intelligence, pages 1–16. Springer-Verlag, 2000.

[9] Martin Fowler. *Analysis Patterns, Reusable Object Models.* Addison Wesley, 1997.

[10] Schreiber G., Akkermans H., Anjewierden A., de Hoog R., Shadbolt N., Van de Velde W., and Wielinga B. *Knowledge Engineering and Managment, The CommonKADS Methodology.* Massachusetts Institute of Technology, 2000.

[11] I. Horrocks and U. Sattler. Ontology Reasoning in the SHOQ(D) Description Logic. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence*, 2001.

[12] Carsten Lutz. Adding numbers to the $\mathcal{SHIQ}$ description logic—First results. In *Proceedings of the Eighth International Conference on Principles of Knowledge Representation and Reasoning (KR2002).* Morgan Kaufman, 2002. To appear.

[13] Van Der Straeten Ragnhild. Using description logic in object-oriented software development. In *Proceedings of the International Description Logic Workshop 2002 (DL2002)*, 2002. To appear.