Using DLs to reason on UML class diagrams

Daniela Berardi
Dipartimento di Informatica e Sistemistica
Università di Roma "La Sapienza"
Via Salaria 113, 00198 Roma, Italy
Email: berardi@dis.uniroma1.it

Abstract

UML is the most widely accepted formalism for the analysis and design of software and one of its most important components are UML class diagrams. In this paper we discuss how to encode UML class diagrams in the Description Logics $\mathcal{DLR}_{ifd}$ and $\mathcal{ALCQI}$: the first *fully captures* the semantics of UML class diagrams, while the second is directly supported by state-of-the-art DL reasoning systems. We also show some results obtained by reasoning on UML class diagrams of industrial interest.

1 Introduction

UML is the most widely accepted formalism for the analysis and design of software. One of its most important components are UML class diagrams, which model the static relationships that hold between the objects of the domain of interest, in terms of classes and associations between them. During the design phase, it is highly desirable to be able to detect relevant formal properties of the diagram, such as inconsistencies and redundancies. This requires a formalization of UML class diagrams and several types of formalizations have already been proposed in the UML literature [12, 13, 14, 11].

Description Logics (DLs) [1] are known to be able to capture several conceptual data models [3, 10, 4, 9, 16], included UML class diagrams [5, 2, 6]. This allows for exploiting DL reasoning services for implementing various forms of reasoning on such diagrams.

In this paper we discuss encodings of UML class diagrams into two DLs, namely $\mathcal{DLR}_{ifd}$ [8, 7], i.e., $\mathcal{DLR}$ with identification constraints and functional dependencies, and $\mathcal{ALCQI}$. The semantics of $\mathcal{DLR}_{ifd}$ *fully captures* UML class diagrams, while $\mathcal{ALCQI}$ is the DL directly supported by state-of-the-art DL reasoning systems [22, 17], and thus facilitates the use of such systems as the core engines for advanced UML CASE tools.

We experimented such encodings by exploiting the reasoning services offered by state-of-the-art automated reasoning systems, within an ongoing project in collaboration with IBM Tivoli, obtaining very encouraging results on industrial relevant diagrams.

The rest of the paper is organized as follows. In Section 2 we give an overview of UML class diagrams and for each construct we provide a $\mathcal{DLR}_{ifd}$-based formalization. In Section 3 we discuss a formalization of UML class diagrams based on $\mathcal{ALCQI}$ and show how and why we can pass from the first to the second one. In Section 4 we present our experiments and discuss them. Section 5 concludes the paper.

2 Representing UML class diagrams

In this section, we briefly illustrate UML class diagrams and for each construct, we provide an encoding in terms of $\mathcal{DLR}_{ifd}$. We refer the reader to [8, 7] for the syntax and semantics of this DL. Scope of this section is to show that $\mathcal{DLR}_{ifd}$ *fully captures* the semantics of UML class diagrams for the conceptual perspective [15].

Classes A *class* in an UML class diagram denotes a *set of objects* with common features, hence it can be represented in $\mathcal{DLR}_{ifd}$ by a concept, since also $\mathcal{DLR}_{ifd}$ concepts denote *sets of objects*.

An *attribute* a of type[1] T for a class C associates to each instance of C a set of instances of T[2]. It can be captured in $\mathcal{DLR}_{ifd}$ by means of a binary relation a between instances of C and instances of T and by the following assertion:

$$C \sqsubseteq \forall[1](a \Rightarrow (2:T)),$$

stating that each relation a having C as first component, has T as second component. An optional multiplicity $[i..j]$ for a specifies that a associates to each instance of C at least i and most j instances of T. When the multiplicity is missing, [1..1] is assumed. Multiplicity can be naturally captured by number restrictions, as follows:

$$C \sqsubseteq (\geq i\,[1]a) \sqcap (\leq j\,[1]a).$$

An *operation* $f(P_1, \ldots, P_m) : (R_1, \ldots, R_n)$ of a class C is a *function* that associates to an m-tuple of parameters belonging to the classes $P_1, \ldots, P_m$, respectively, an n-tuple of return values belonging to the classes $R_1, \ldots, R_n$, respectively. Recalling that $\mathcal{DLR}_{ifd}$ has the ability of representing n-ary relations

[1]For simplicity, we consider types as classes, without distinguishing between classes and domains, such as integers, reals, ...

[2]Note that UML allows one to define two classes having the same attribute, possibly of different types.

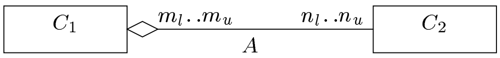

Figure 1: Aggregation in UML

and functional dependencies on them, we capture the operation above by means of a relation named $\mathtt{op}_{f(P_1,\ldots,P_m):(R_1,\ldots,R_n)}$ of arity $m+n+1$, among instances of the concepts $C, P_1, \ldots, P_m, R_1, \ldots, R_n$. On such a relation we impose:

- that parameters and return values have correct types:

$$\begin{aligned} C \ \sqsubseteq \ & \forall[1](\mathtt{op}_{f(P_1,\ldots,P_m):(R_1,\ldots,R_n)} \Rightarrow \\ & ((2\!:\!P_1) \sqcap \cdots \sqcap (m+1\!:\!P_m) \sqcap (m+2\!:\!R_1) \sqcap \cdots \sqcap (m+n+1\!:\!R_n))); \end{aligned}$$

- functional dependencies from the class the operation belongs to and the input parameters, to each return value:

$$\begin{gathered} (\mathbf{fd}\ \mathtt{op}_{f(P_1,\ldots,P_m):(R_1,\ldots,R_n)}\ 1,\ldots,m+1 \rightarrow m+2), \\ \ldots, \\ (\mathbf{fd}\ \mathtt{op}_{f(P_1,\ldots,P_m):(R_1,\ldots,R_n)}\ 1,\ldots,m+1 \rightarrow m+n+1), \end{gathered}$$

  such assertions state that each return value is uniquely determined by the object of invocation and the input parameters.

Observe that the formalization of operations in $\mathcal{DLR}_{ifd}$ allows one to have operations with the same name or even with the same signature in two different classes. In particular, following UML specifications, our framework allows for *overloading* of methods, that takes place between two or more functions having the same name but different signatures. *Overriding* requires that two methods have the same name and signature, but behave in different ways. UML class diagrams for the conceptual perspective do not specify bodies of methods but only their signature, hence overriding becomes immaterial and so does in our representation.

Aggregations An *aggregation* in UML is a binary relation between the instances of two classes, denoting a generic form of part-whole relationship: aggregation A in Figure 1 indicates that the instances of the (containing) class C_1 have components that are instances of the (contained) class C_2. This is formalized by means of a binary relation A together with the assertion

$$A \ \sqsubseteq \ (1\!:\!C_1) \sqcap (2\!:\!C_2).$$

The distinction between the contained class and the containing class is not lost, since we can use the convention that the first argument of the relation is the containing class.

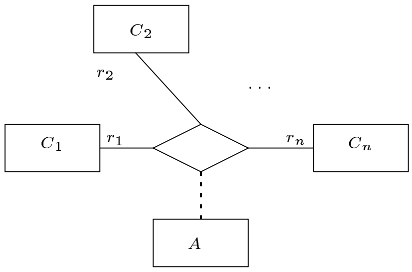

Figure 2: Association in UML

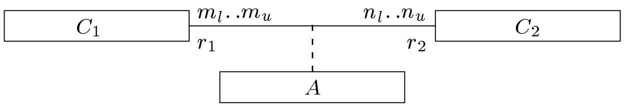

Figure 3: Binary association in UML

The multiplicity of an aggregation can be easily expressed. For example, the multiplicities shown in Figure 1 are formalized in $\mathcal{DLR}_{ifd}$ as:

$$\begin{array}{rcl} C_1 & \sqsubseteq & (\geq n_l\,[1]A) \sqcap (\leq n_u\,[1]A), \\ C_2 & \sqsubseteq & (\geq m_l\,[2]A) \sqcap (\leq m_u\,[2]A). \end{array}$$

Associations An *association* in UML, graphically rendered as in Figure 2, is a relation between the instances of two or more classes. An association often has a related *association class* that describes properties of the association such as attributes, operations, etc.

We model in $\mathcal{DLR}_{ifd}$ an n-ary association A between classes $C_1, \ldots C_n$, that has not a related association class, by means of an n-ary relation A, and the assertion

$$A \sqsubseteq (1\!:\!C_1) \sqcap \ldots \sqcap (n\!:\!C_n),$$

stating that A has C_i as i-th component, for $i = 1, \ldots, n$.

If A has a related association class, we formalize it in a reified way: we introduce a concept A and n *binary* relations $r_1, \ldots, r_n$, one for each component of the association A. Each binary relation r_i has A as its first component and as C_i its second component. Then we introduce the following assertions:

$$\begin{array}{rcl} A & \sqsubseteq & \exists[1]r_1 \sqcap (\leq 1\,[1]r_1) \sqcap \forall[1](r_1 \Rightarrow (2\!:\!C_1)) \sqcap \ldots \sqcap \\ & & \exists[1]r_n \sqcap (\leq 1\,[1]r_n) \sqcap \forall[1](r_n \Rightarrow (2\!:\!C_n)) \end{array}$$

$$(\mathbf{id}\ A\ [1]r_1, \ldots, [1]r_n).$$

The first assertion states that the concept A must have all components $r_1, \ldots, r_n$ of the association A and that each such component is single-valued; it also specifies the class each component has to belong to. The second assertion specifies

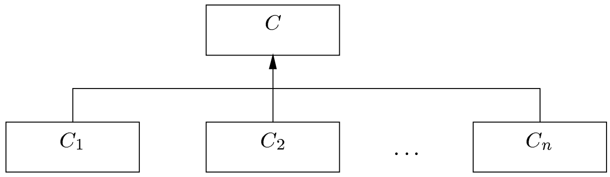

Figure 4: A class hierarchy in UML

that each instance of A represents a *distinct* tuple in $C_1 \times \cdots \times C_n$. We remind that an important feature of $\mathcal{DLR}_{ifd}$ is the ability of capturing identification constraints on concepts.

For a binary UML association[3], we can easily represent multiplicities by imposing suitable number restrictions on the relations modeling the components of the association. The multiplicities shown in Figure 3 are captured by:

$$\begin{array}{lcl} C_1 & \sqsubseteq & (\geq n_l\,[2](r_1 \sqcap (1:A))) \sqcap (\leq n_u\,[2](r_1 \sqcap (1:A))) \\ C_2 & \sqsubseteq & (\geq m_l\,[2](r_2 \sqcap (1:A))) \sqcap (\leq m_u\,[2](r_2 \sqcap (1:A))). \end{array}$$

Generalization In UML, a *generalization* between a parent class and a child class, as shown in Figure 4, is used to specify that each instance of the child class is also an instance of the parent class. The instances of the child class inherit the properties of the parent class, and usually they satisfy additional properties that do not hold for the parent class.

Generalization is naturally supported in $\mathcal{DLR}_{ifd}$. We can express that an UML class C generalizes n classes $C_1, \ldots, C_n$, by the set of assertions

$$C_i \ \sqsubseteq\ C, \qquad \text{for each } i \in \{1, \ldots, n\}.$$

Inheritance between $\mathcal{DLR}_{ifd}$ concepts works exactly as inheritance between UML classes, as a consequence of the semantics of inclusion assertions, which is based on sub-setting: given an assertion $C_1 \sqsubseteq C_2$, every tuple in a relation having C_2 as i-th argument type may have as i-th component an instance of C_1, which is in fact also an instance of C_2. Hence, each attribute or operation of C_2, and each aggregation and association involving C_2 is correctly inherited by C_1. The above formalization also captures directly inheritance among association classes, which are treated exactly as all other classes, and multiple inheritance between classes (including association classes).

In UML, one can impose covering or mutual disjointness between classes. If the superclass C is a *covering* of the subclasses $C_1, \ldots, C_n$, we include the additional assertion

$$C \ \sqsubseteq\ C_1 \sqcup \cdots \sqcup C_n.$$

[3]In UML multiplicities are look-across cardinality constraints [24]. This makes their use in non-binary associations difficult w.r.t. both modeling and reasoning.

For each pair of subclasses C_i and C_j that are *mutually disjoint*, we include the assertion

$$C_i \sqsubseteq \neg C_j.$$

Constraints In UML it is possible to add *constraints*, in order to express application semantics which cannot be expressed by other constructs of UML class diagrams. One can exploit the expressive power of $\mathcal{DLR}_{ifd}$ to formalize several types of constraints (although not all those that OCL [23], i.e. first order logic, can), that can be taken fully into account when reasoning on class diagrams.

3 Representing UML class diagrams in $\mathcal{ALCQI}$

$\mathcal{DLR}_{ifd}$ *fully* captures the semantics of UML class diagrams (except for general OCL constructs). Functional dependencies and identification constraints play a special role since allow one to impose that each instance of a concept representing an n-ary association A (with a related association class) is univocally identified by each first component of the n roles r_i and, similarly, that each return value of an operation f of arity m belonging to class C is determined once the instances of the class and of the input parameters are given. Current state-of-the-art DL-based reasoning systems do not support functional dependencies and identification constraints yet: these require advanced forms of reasoning on generalized ABoxes that are not implemented yet [8]. In [8] it is shown that for a KB without ABox, logical implication of inclusion assertions (note, not of functional dependency and identification constraint assertions) can be verified without considering functional dependencies and identification constraints at all. This is because a $\mathcal{DLR}_{ifd}$ TBox has the tree model property [7], i.e., if a TBox is satisfiable, it admits a model having a tree structure, and since in such models no tuples exist having more than one component in common, identification constraints and functional dependencies are trivially satisfied. We are exactly in this setting. Hence, if we limit to ask queries not involving either functional dependencies or identification constraints, we can drop them, obtaining a KB in a DL named $\mathcal{DLR}$. $\mathcal{DLR}$ KBs can be translated into $\mathcal{ALCQI}$ KBs[4] and we can exploit DL reasoning systems to deduce relevant properties of UML class diagrams. However, this brings about a disruptive mixture of inverse roles with terminological cycles involving existentials, and with functional restrictions combined with existential restrictions. In fact, in [2], we found out that state-of-the-art reasoners have difficulties in classifying KBs obtained in such a way[5].

[4] Actually, in the variants of $\mathcal{ALCQI}$ accepted by the reasoners.

[5] In fact, techniques that are currently being developed to deal with complex KBs (see [21]) can be of help.

To overcome this problem, we propose a direct encoding of UML class diagrams in $\mathcal{ALCQI}$, in order to obtain KBs on which reasoning services are easier for the reasoning systems. In particular, the encoding of UML class diagram constructs is as follows:

- *classes* are modeled by $\mathcal{ALCQI}$ concepts; *attribute* a of type T for class C is captured by a binary relation a and by the assertion $C \sqsubseteq (\geq i\, a) \sqcap (\leq j\, a)$; if the attribute has associated a multiplicity $[i..j]$, this is captured by $C \sqsubseteq (\geq i\, a) \sqcap (\leq j\, a)$; *operation* $f(P_1,\ldots,P_m) : (R_1,\ldots,R_n)$ of a class C is represented in a reified way, i.e., by introducing an atomic concept $\mathtt{op}_{f(P_1,\ldots,P_m):(R_1,\ldots,R_n)}$, $n+m+1$ roles $r_1,\ldots,r_{n+m+1}$ and the following assertion: $\mathtt{op}_{f(P_1,\ldots,P_m):(R_1,\ldots,R_n)} \sqsubseteq \forall r_1.C \sqcap \exists r_1 \sqcap (\leq 1\, r_1) \sqcap \forall r_2.P_1 \sqcap \exists r_2 \sqcap (\leq 1\, r_2) \sqcap \cdots \sqcap \forall r_{m+n+1}.R_n \sqcap \exists r_{n+m+1} \sqcap (\leq 1\, r_{m+n+1})$;
- *aggregation* A in Figure 1 between the containing class C_1 and the contained class C_2 is modeled by $\top \sqsubseteq \forall A.C_2 \sqcap \forall A^-.C_1$; the multiplicity constraints shown there are captured by $C_1 \sqsubseteq (\geq n_l\, A) \sqcap (\leq n_u\, A)$ and $C_2 \sqsubseteq (\geq m_l\, A^-) \sqcap (\leq m_u\, A^-)$;
- *n-ary association* A between classes $C_1,\ldots C_n$, shown in Figure 2, is captured by reifying the association and by the assertion $A \sqsubseteq (\leq 1\, r_1) \sqcap \cdots \sqcap (\leq 1\, r_n) \sqcap \exists r_1.C_1 \sqcap \cdots \sqcap \exists r_n.C_n$, both if it has and it has not a related association class; the multiplicity constraints shown in Figure 3 are represented by $C_1 \sqsubseteq (\geq n_l\, r_1^-.A) \sqcap (\leq n_u\, r_1^-.A)$ and $C_2 \sqsubseteq (\geq m_l\, r_2^-.A) \sqcap (\leq m_u\, r_2^-.A)$.

It can be shown that such encoding is equivalent to the $\mathcal{DLR}$ one. In particular, the ability of correctly representing an n-ary relation through reification, i.e. with a concept plus one role for each relation component, is again granted by the tree model property.

4 Experimentation

We performed several experiments on UML class diagrams. In this section we report results on (successful) classification of diagrams, available on the site `http://www.dis.uniroma1.it/~berardi/uml2dl`. The Table in Figure 5 gives an idea of how easier are the KBs obtained from the encoding above, for the reasoners. In particular, we have used the two systems FaCT [19, 22, 20][6] (the executable $\mathcal{SHIQ}$ reasoner (shiq-app.exe) contained in the CORBA-FaCT distribution v.3.1, excluding the CORBA interface) and Racer [18, 17][7] (v.1-6-7). We have run all our experiments on a Pentium III biprocessor, 866 Mhz, 512MB of RAM and OS Windows 2000 Professional. The first set of columns

[6] Available at `http://www.cs.man.ac.uk/~horrocks/FaCT`.

[7] Available at `http://kogs-www.informatik.uni-hamburg.de/~race`.

UML class diagrams	$\mathcal{DLR}$-based encoding		$\mathcal{ALCQI}$-based encoding	
	FaCT	Racer	FaCT	Racer
Restaurant	yes	no	yes	yes
Soccer	no	no	yes	yes
Library	yes	no	yes	yes
Hospital	yes	no	yes	yes
School-exam	no	no	yes	yes
Video library	no	no	yes	yes
Workshop	no	no	yes	yes

Figure 5: Successful classification of the presented KBs.

in the table reports the results of the classification of the KBs obtained from a $\mathcal{DLR}$-based encoding of UML class diagrams and the second one reports the results of the classification of the KBs obtained following the $\mathcal{ALCQI}$-based encoding. Besides, "yes" means that the reasoners can classify the KBs, "no" that they cannot, because they run out of resources. As one can see, both reasoners can classify all the presented KBs obtained with the $\mathcal{ALCQI}$-based encoding, whereas this does not happen with the $\mathcal{DLR}$-based encoding.

We also performed several experiments on UML class diagrams as part of an ongoing project in collaboration with IBM Tivoli. Scope of this project is to study how to extract intensional knowledge from Common Information Model, that can be useful to support the management of an elaboration system. In particular, we want to exploit reasoning systems to derive such knowledge, from an arbitrarily complex UML class diagram.

Common Information Model (CIM) is a model based on UML with the purpose of providing a rigorous approach for modelling systems and networks using the object-oriented paradigm. CIM has a *Meta Schema* that specifies how other schemas can be constructed, in order to form the basis for a sort of vocabulary for analyzing and describing managed systems. CIM offers two main conceptual layers, that form the *CIM Schema*: the *Core Model*, that captures basic notions, common to all areas of management, and the *Common Model*, that expresses concepts related to specific management areas (e.g., device, networks, systems, etc.). In our experiments, we focused on subschemas of CIM Schema v.26[8].

In our experiments we considered both the $\mathcal{DLR}$-based encoding and the $\mathcal{ALCQI}$-based one. FaCT and Racer can classify *all* the KBs presented in a few seconds. This is due to the fact that they have few multiplicity constraints on associations and aggregations, which leads to few terminological cycles. As we expected, with the latter encoding, reasoners take less than with the former one, since the KBs obtained are much simpler.

[8]They are available on the page http://www.dmtf.org/standards/cim_schema_v26.php.

5 Conclusions

In this paper, we have discussed two encodings of UML class diagrams in terms of $\mathcal{DLR}_{ifd}$ and of $\mathcal{ALCQI}$. The $\mathcal{DLR}_{ifd}$ encoding allows us to capture exactly the semantics of UML class diagram constructs, since $\mathcal{DLR}_{ifd}$ can represent n-ary relations, functional dependencies on relations and identification constraints on concepts. By dropping functional dependencies and identification constraints, that cannot be dealt with yet, one can use the state-of-the-art reasoners. However, we noticed that they have still difficulties in reasoning on KBs exploiting the $\mathcal{DLR}$-based encoding of complex UML class diagrams. Hence we have devised a direct encoding into $\mathcal{ALCQI}$, which is the DL adopted by state-of-the-art DL-based reasoning systems. The experiments we have reported show that the KBs obtained in this case are easily classified by the reasoners. The use of $\mathcal{DLR}_{ifd}$ (but also of $\mathcal{DLR}$) remains a challenge for the future DL-based systems.

Acknowledgments. The author would like to thank Giuseppe De Giacomo and Diego Calvanese for their support during the writing of this paper.

References

[1] F. Baader, D. McGuinness, D. Nardi, and P. F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation and Applications*. Cambridge University Press, 2002. To appear.

[2] D. Berardi, D. Calvanese, and G. De Giacomo. Reasoning on UML class diagrams using description logic based systems. In *Proc. of the KI'2001 Workshop on Applications of Description Logics*. CEUR Electronic Workshop Proceedings, `http://ceur-ws.org/Vol-44/`, 2001.

[3] S. Bergamaschi and C. Sartori. On taxonomic reasoning in conceptual design. *ACM Trans. on Database Systems*, 17(3):385–422, 1992.

[4] A. Borgida. Description logics in data management. *IEEE Trans. on Knowledge and Data Engineering*, 7(5):671–682, 1995.

[5] A. Calì, D. Calvanese, G. De Giacomo, and M. 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.

[6] A. Calì, D. Calvanese, G. De Giacomo, and M. Lenzerini. A formal framework for reasoning on uml class diagrams. In *Proc. of the 13th Int. Sym. on Methodologies for Intelligent Systems (ISMIS 2002)*, pages 503–513, 2002.

[7] D. Calvanese, G. De Giacomo, and M. Lenzerini. On the decidability of query containment under constraints. In *Proc. of PODS'98*, pages 149–158, 1998.

[8] D. Calvanese, G. De Giacomo, and M. Lenzerini. Identification constraints and functional dependencies in description logics. In *Proc. of IJCAI 2001*, pages 155–160, 2001.

[9] D. Calvanese, M. Lenzerini, and D. Nardi. Unifying class-based representation formalisms. *J. of Artificial Intelligence Research*, 11:199–240, 1999.

[10] T. Catarci and M. Lenzerini. Representing and using interschema knowledge in cooperative information systems. *J. of Intelligent and Cooperative Information Systems*, 2(4):375–398, 1993.

[11] T. Clark and A. S. Evans. Foundations of the Unified Modeling Language. In *Proc. of the 2nd Northern Formal Methods Workshop*. Springer-Verlag, 1997.

[12] A. Evans, R. France, K. Lano, and B. Rumpe. The UML as a formal modeling notation. In *Proc. of the OOPSLA'97 Workshop on Object-oriented Behavioral Semantics*, pages 75–81, 1997.

[13] A. Evans, R. France, K. Lano, and B. Rumpe. Meta-modelling semantics of UML. In *Behavioural Specifications for Businesses and Systems*, chapter 2. Kluwer Academic Publisher, 1999.

[14] A. S. Evans. Reasoning with UML class diagrams. In *Second IEEE Workshop on Industrial Strength Formal Specification Techniques (WIFT'98)*. IEEE CS Press, 1998.

[15] M. Fowler and K. Scott. *UML Distilled – Applying the Standard Object Modeling Laguage*. Addison Wesley Publ. Co., Reading, Massachussetts, 1997.

[16] E. Franconi and G. Ng. The i.com tool for intelligent conceptual modeling. In *Proc. of KRDB 2000*, pages 45–53. CEUR Electronic Workshop Proceedings, `http://ceur-ws.org/Vol-29/`, 2000.

[17] V. Haarslev and R. Möller. High performance reasoning with very large knowledge bases: A practical case study. In *Proc. of IJCAI 2001*, pages 161–168, 2001.

[18] V. Haarslev and R. Möller. RACER system description. In *Proc. of IJCAR 2001*, volume 2083 of *LNAI*, pages 701–705. Springer-Verlag, 2001.

[19] I. Horrocks. Using an expressive description logic: FaCT or fiction? In *Proc. of KR'98*, pages 636–647, 1998.

[20] I. Horrocks and P. F. Patel-Schneider. Optimizing description logic subsumption. *J. of Log. and Comp.*, 9(3):267–293, 1999.

[21] I. Horrocks and U. Sattler. Optimised reasoning for $\mathcal{SHIQ}$. In *Proc. of the 15th Eur. Conf. on Artificial Intelligence (ECAI 2002)*, July 2002.

[22] I. Horrocks, U. Sattler, and S. Tobies. Practical reasoning for expressive description logics. In *Proc. of LPAR'99*, number 1705 in LNAI, pages 161–180. Springer-Verlag, 1999.

[23] J. Rumbaugh, I. Jacobson, and G. Booch. *The Unified Modeling Language Reference Manual.* Addison Wesley Publ. Co., Reading, Massachussetts, 1998.

[24] B. Thalheim. Fundamentals of cardinality constraints. In *Proc. of ER'92*, pages 7–23. Springer-Verlag, 1992.