Reasoning on UML Class Diagrams using Description Logic Based Systems

Daniela Berardi, Diego Calvanese, Giuseppe De Giacomo
Dipartimento di Informatica e Sistemistica
Università di Roma "La Sapienza"
Via Salaria 113, 00198 Roma, Italy
lastname@dis.uniroma1.it

Abstract

In this paper we study how automated reasoning systems based on Description Logics (DLs) can be used for reasoning about UML class diagrams. The ability of reasoning automatically on UML class diagrams makes it possible to provide computer aided support during the application design phase in order to automatically detect relevant properties, such as inconsistencies and redundancies. We show that UML class diagrams can be formalized as knowledge bases expressed in the DL $\mathcal{DLR}$. $\mathcal{DLR}$ knowledge bases can be translated into knowledge bases expressed in the variants of $\mathcal{ALCQI}$ accepted by state-of-the-art DL-based systems. Hence, in principle, the reasoning capabilities of such systems can be used to reason on UML class diagrams. However, we report some experiments indicating that state-of-the-art systems have still difficulty in dealing with the resulting knowledge bases.

1 Introduction

The *Unified Modeling Language* (UML) is the de facto standard formalism for object-oriented modeling [1, 11]. There is a vast consensus on the need for a precise semantics for UML [9, 14], in particular for UML class diagrams. Indeed, several kinds of formalizations of UML class diagrams have been proposed in the literature [8, 9, 10, 7]. Many of them have been proved very useful with respect to the task of establishing a common understanding of the formal meaning of UML constructs. However, to the best of our knowledge, none of them has the explicit goal of building a solid basis for allowing automated reasoning techniques, based on algorithms that are sound and complete wrt the semantics, to be applicable to UML class diagrams.

We are interested in exploiting the research on Description Logics (DLs), which are decidable logics tailored towards class based knowledge representation, to carry out various forms of reasoning on UML class diagrams, so as to provide support during the specification phase of software development. Recently the research on DLs has resulted in a number of automated reasoning systems [15, 16, 17, 12, 13], that have been successfully tested in various application domains (see e.g., [19, 20, 18]). Such systems are candidates to form the core reasoning engine for advanced UML CASE tools.

In this paper, we illustrate a formalization of UML class diagrams in terms of DLs [2]. In particular, we show how UML class diagrams can be captured by knowledge bases expressed in the DL $\mathcal{DLR}$ [4, 3]. This logic is particularly well tailored towards the high expressiveness of UML information structuring mechanisms, and allows one to easily model important additional properties, such as disjointness of classes, or partitions of classes into subclasses, that are typically specified by means of constraints in UML class diagrams. $\mathcal{DLR}$ assertions can be translated into $\mathcal{ALCQI}$ assertions. Since variants of the latter are accepted by state-of-the-art DL-based reasoning systems, in principle, we can exploit such systems to reason about UML class diagrams. However, in spite of the fact that such systems have shown to perform nicely in several context, we report in this paper some experiments indicating that they still have serious efficiency problems when dealing with UML class diagrams.

The rest of the paper is organized as follows. In Section 2 we give a brief overview of the Description Logic $\mathcal{DLR}$. In Section 3 we show how UML class diagrams can be formalized in $\mathcal{DLR}$. In Section 4 we discuss the use of DL-based reasoning systems, namely FaCT [17] and Racer [13], for reasoning about UML class diagrams, and show some results of our experimentation with such systems. Section 5 concludes the paper. In the appendix, we show the UML class diagrams used in the reported experiments.

2 The Description Logic $\mathcal{DLR}$

The basic elements of $\mathcal{DLR}$ [4, 3] are *concepts* and *n-ary relations*. We assume to deal with a finite set of atomic relations and atomic concepts, denoted by P and A, respectively. Arbitrary relations (of given arity between 2 and n_{max}), denoted by R, and arbitrary concepts, denoted by C, are built according to the following syntax:

$$\begin{array}{rcl} R & ::= & \top_n \mid P \mid (i/n : C) \mid \neg R \mid R_1 \sqcap R_2 \\ C & ::= & \top_1 \mid A \mid \neg C \mid C_1 \sqcap C_2 \mid (\leq k\, [i] R) \end{array}$$

where i denotes a component of a relation, i.e., an integer between 1 and n_{max}, n denotes the *arity* of a relation, i.e., an integer between 2 and n_{max}, and k denotes

$$
\begin{array}{rcl@{\qquad}rcl}
\top_n^{\mathcal{I}} & \subseteq & (\Delta^{\mathcal{I}})^n & \top_1^{\mathcal{I}} & = & \Delta^{\mathcal{I}} \\
P^{\mathcal{I}} & \subseteq & \top_n^{\mathcal{I}} & A^{\mathcal{I}} & \subseteq & \Delta^{\mathcal{I}} \\
(i/n:C)^{\mathcal{I}} & = & \{t \in \top_n^{\mathcal{I}} \mid t[i] \in C^{\mathcal{I}}\} & (\neg C)^{\mathcal{I}} & = & \Delta^{\mathcal{I}} \setminus C^{\mathcal{I}} \\
(\neg R)^{\mathcal{I}} & = & \top_n^{\mathcal{I}} \setminus R^{\mathcal{I}} & (C_1 \sqcap C_2)^{\mathcal{I}} & = & C_1^{\mathcal{I}} \cap C_2^{\mathcal{I}} \\
(R_1 \sqcap R_2)^{\mathcal{I}} & = & R_1^{\mathcal{I}} \cap R_2^{\mathcal{I}} & (\leq k\,[i]R)^{\mathcal{I}} & = & \{a \in \Delta^{\mathcal{I}} \mid \sharp\{t \in R_1^{\mathcal{I}} \mid t[i] = a\} \leq k\}
\end{array}
$$

Figure 1: Semantic rules for $\mathcal{DLR}$ (P, R, R_1, and R_2 have arity n)

a non-negative integer. We consider only concepts and relations that are *well-typed*, which means that (i) only relations of the same arity n are combined to form expressions of type $R_1 \sqcap R_2$ (which inherit the arity n), and (ii) $i \leq n$ whenever i denotes a component of a relation of arity n.

We also make use of the following abbreviations: $C_1 \sqcup C_2$ for $\neg(\neg C_1 \sqcap \neg C_2)$, $C_1 \Rightarrow C_2$ for $\neg C_1 \sqcup C_2$, $(\geq k\,[i]R)$ for $\neg(\leq k{-}1\,[i]R)$, $\exists[i]R$ for $(\geq 1\,[i]R)$, $\forall[i]R$ for $\neg\exists[i]\neg R$. Moreover, we abbreviate $(i/n:C)$ with $(i:C)$, when n is clear from the context.

A $\mathcal{DLR}$ *knowledge base* (KB) is constituted by a finite set of *inclusion assertions*, where each assertion has one of the forms:

$$R_1 \sqsubseteq R_2 \qquad\qquad C_1 \sqsubseteq C_2$$

with R_1 and R_2 of the same arity.[1]

The semantics of $\mathcal{DLR}$ is specified through the notion of interpretation. An *interpretation* $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ of a $\mathcal{DLR}$ KB $\mathcal{K}$ is constituted by an *interpretation domain* $\Delta^{\mathcal{I}}$ and an *interpretation function* $\cdot^{\mathcal{I}}$ that assigns to each concept C a subset $C^{\mathcal{I}}$ of $\Delta^{\mathcal{I}}$ and to each relation R of arity n a subset $R^{\mathcal{I}}$ of $(\Delta^{\mathcal{I}})^n$, such that the conditions in Figure 1 are satisfied (in the figure, $t[i]$ denotes the i-th component of tuple t). We observe that $\top_1$ denotes the interpretation domain, while $\top_n$, for $n > 1$, does *not* denote the n-Cartesian product of the domain, but only a subset of it that covers all relations of arity n. It follows, from this property, that the "$\neg$" constructor on relations is used to express difference of relations, rather than complement.

To specify the semantics of a KB we first define when an interpretation satisfies an assertion as follows: An interpretation $\mathcal{I}$ *satisfies* an inclusion assertion $R_1 \sqsubseteq R_2$ (resp. $C_1 \sqsubseteq C_2$) if $R_1^{\mathcal{I}} \subseteq R_2^{\mathcal{I}}$ (resp. $C_1^{\mathcal{I}} \subseteq C_2^{\mathcal{I}}$). An interpretation that satisfies all assertions in a KB $\mathcal{K}$ is called a *model* of $\mathcal{K}$.

Several reasoning services are applicable to $\mathcal{DLR}$ KBs. The most important ones are KB satisfiability and logical implication. A KB $\mathcal{K}$ is *satisfiable* if there

[1] $\mathcal{DLR}$ knowledge bases may also include *identification-constraints* that allow one to force instances of concepts or relations to be uniquely identified through suitable mechanisms (see [4] for details). Interestingly, however, such additional constraints play no role in checking knowledge base satisfiability or logical implication of inclusion assertions. For this reason in this paper we do not consider them.

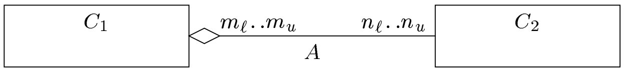

Figure 2: Aggregation in UML

exists a model of $\mathcal{K}$. An inclusion assertion α is *logically implied* by $\mathcal{K}$ if all models of $\mathcal{K}$ satisfy α. One can easily verify that logical implication and KB (un)satisfiability are mutually reducible.

One of the distinguishing features of $\mathcal{DLR}$ is that it is equipped with reasoning algorithms that are sound and complete wrt to the semantics. Such algorithms allow one to decide all the above reasoning tasks in deterministic exponential time [4, 3].

3 Representing UML class diagrams

We concentrate on UML class diagrams for the conceptual perspective. Hence, we do not deal with those features that are relevant for the implementation perspective, such as public, protected, and private qualifiers for methods and attributes.

Classes A *class* in an UML class diagram denotes a *set of objects* with common features, hence it can be represented by a $\mathcal{DLR}$ concept. This follows naturally from the fact that both UML classes and $\mathcal{DLR}$ concepts denote *sets of objects*. Attributes and operations of classes can be easily represented by means of $\mathcal{DLR}$-relations [2].

Relationships between classes come in two forms in UML: aggregations, denoting part-whole relationships, and associations, denoting general relationships between two or more classes.

Aggregations An *aggregation* in UML, graphically rendered as in Figure 2, is a binary relation between the instances of two classes, denoting a generic form of part-whole relationship, i.e., a relationship that specifies that each instance of a class is made up of a set of instances of another class. An aggregation A, saying that instances of the class C_1 have components that are instances of the class C_2, is formalized in $\mathcal{DLR}$ by means of a binary relation A together with the following assertion:

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

Note that the distinction between the contained class and the containing class is not lost. Indeed, we simply use the following convention: *the first argument of the relation is the containing class.* The multiplicity of an aggregation can be

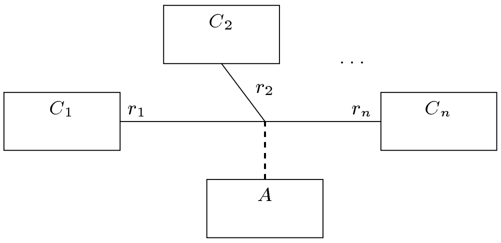

Figure 3: Association in UML

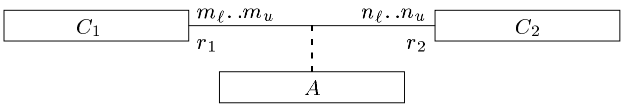

Figure 4: Binary association in UML

easily expressed in $\mathcal{DLR}$. For example, the multiplicities shown in Figure 2 are formalized by means of the assertions:

$$\begin{array}{lll} C_1 & \sqsubseteq & (\geq n_\ell\,[1]A) \sqcap (\leq n_u\,[1]A) \\ C_2 & \sqsubseteq & (\geq m_\ell\,[2]A) \sqcap (\leq m_u\,[2]A) \end{array}$$

Associations An *association* in UML, graphically rendered as in Figure 3, 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.

Since associations have often a related association class, we formalize associations in $\mathcal{DLR}$ by reifying each association A into a $\mathcal{DLR}$ concept A with suitable properties. We represent an association among n classes $C_1, \ldots, C_n$, as shown in Figure 3, by introducing 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 C_i as its first component and A as its second component. Then we introduce the following assertion:

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

where $\exists[1]r_i$, with $i \in \{1, \ldots, n\}$, specifies that the concept A must have all components $r_1, \ldots, r_n$ of the association A, $(\leq 1\,[1]r_i)$ specifies that each such component is single-valued, and $\forall[1](r_i \Rightarrow (2\!:\!C_i))$ specifies the class each component has to belong to.[2]

[2]In addition, we would need an identification constraint saying that the relations $r_1, \ldots, r_n$

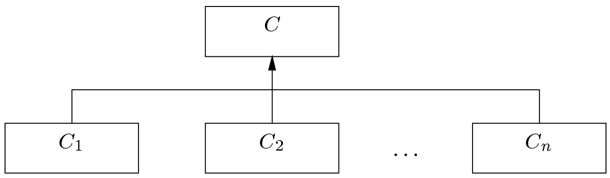

Figure 5: A class hierarchy in UML

For a binary UML association, we can easily represent multiplicities by imposing suitable number restrictions on the $\mathcal{DLR}$ relations modeling the components of the association. The multiplicities shown in Figure 4 are captured as follows:

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

Generalization In UML one can use *generalization* between a parent class and a child class to specify that each instance of the child class is also an instance of the parent class. Hence, the instances of the child class inherit the properties of the parent class, but typically they satisfy additional properties that do not hold for the parent class.

Generalization is naturally supported in $\mathcal{DLR}$. If an UML class C_2 generalizes a class C_1, we can express this by the $\mathcal{DLR}$ assertion:

$$C_1 \sqsubseteq C_2$$

Inheritance between $\mathcal{DLR}$ concepts works exactly as inheritance between UML classes. This is an obvious consequence of the semantics of inclusion assertions, which is based on subsetting. Indeed, in $\mathcal{DLR}$, 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. As a consequence, in the formalization, each attribute or operation of C_2, and each aggregation and association involving C_2 is correctly inherited by C_1. Observe that the formalization in $\mathcal{DLR}$ 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 group several generalizations, as shown e.g.., in Figure 5, and impose covering or mutual disjointness between classes, if needed. This is captured in $\mathcal{DLR}$ by a set of inclusion assertions, one between each child class

form an identifier of the concept A. However, as mentioned, such a constraint has no impact on reasoning on logical implication or satisfiability of the resulting knowledge base, so we omit it here.

and the parent class:

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

Then if the superclass C is a *covering* of the subclasses $C_1,\ldots,C_n$, we include the additional assertion

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

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

$$C_i \sqsubseteq \neg C_j$$

Constraints In UML it is possible to add information to a class diagram by using *constraints*. In general, constraints are used to express in an informal way information which cannot be expressed by other constructs of UML class diagrams. One can exploit the expressive power of $\mathcal{DLR}$ to formalize several types of constraints that allow one to better represent the application semantics and that are typically not dealt with in a formal way. This allows one to take such constraints fully into account when reasoning on the class diagram.

4 Experiments

We have formalized as $\mathcal{DLR}$ knowledge bases several UML class diagrams. Then we have used state-of-the-art DL-based systems to reason with them, by translating the $\mathcal{DLR}$ knowledge bases into $\mathcal{ALCQI}$ knowledge bases (or more precisely knowledge bases expressed in the variants of $\mathcal{ALCQI}$ accepted by the systems used). In particular, we have used the two systems FaCT [3] (the executable $\mathcal{SHIQ}$ reasoner (shiq-app.exe) contained in the Corba-FaCT distribution v.2.15, excluding the corba interface) and Racer [4] (v.1-5-10). We have run the experiments on a Pentium III biprocessor, 866 Mhz, 512MB of RAM and OS Windows 2000 Professional.

Below we report the results obtained with four rather simple UML class diagrams, shown in the appendix: Restaurant, Library, Soccer, and Hospital, modeling, respectively, a restaurant, a library, a soccer championship and the acceptance procedure in a hospital. The reasoning service we focused on is satisfiability of the class diagram. Observe that all diagrams are obviously satisfiable. The obtained results are shown in Table 1, where:

- complete refers to the original UML class diagrams;

[3] Available at `http://www.cs.man.ac.uk/~horrocks/FaCT`.
[4] Available at `http://kogs-www.informatik.uni-hamburg.de/~race`.

	Restaurant		Hospital		Soccer		Library	
	FaCT	Racer	FaCT	Racer	FaCT	Racer	FaCT	Racer
no mult. constr.	yes	yes	yes	yes	yes	yes	yes	yes
no minimal mult. constr.	yes	no	yes	yes	yes	yes	yes	yes
no maximal mult. constr.	yes	no	yes	no	yes	no	yes	yes
complete	no	no	yes	no	no	no	yes	no

Table 1: Successful classification of the considered UML class diagrams

- no multiplicity const. refers to the class diagrams weakened by removing all multiplicity constraints, i.e., making all multiplicities of the form $0..*$;
- no minimal multiplicity const. refers to the class diagrams weakened by removing minimal multiplicity constraints, thus getting multiplicities of the form $0..*$ or $0..1$;
- no maximal multiplicity const. refers to the class diagrams weakened by removing maximal multiplicity constraints, thus getting multiplicities of the form $0..*$ or $1..*$.

In the table, "yes" indicates that the reasoner could classify the knowledge base corresponding to the UML class diagram, and "no" that the reasoner couldn't classify it because it ran out of resources.

When the reasoners are able to classify a knowledge base (yes in the table), they both take less than 1 minute to perform the classification. When FaCT cannot classify a knowledge base (no in the table), this is because it goes in stack overflow (in about 1 minute on the experiments reported). Observe that the only limit to the stack size is the one imposed by the OS, and FaCT goes in stack overflow whenever the OS can't provide more memory. FaCT memory requests increase quite regularly, until all the available memory is exhausted. As for Racer, when it cannot classify a knowledge base (no in the table, this is because it starts paging, so that all the resources are used to perform memory swaps and the CPU usage decreases greatly. After 1 hour of paging we stopped the reasoner. The only exception to this behaviour is in classifying the knowledge base corresponding to Hospital with no maximal multiplicity constraints, where Racer goes in stack overflow, even setting the stack size to the maximum.

FaCT can classify all knowledge bases corresponding to the class diagrams having no minimal multiplicity constraints and those having no maximal multiplicity constraints, but it can't classify some of those corresponding to the complete class diagrams: namely Soccer and Restaurant, which are characterized by having cycles of associations/aggregations all involving minimal multiplicity constraints in both directions. [5]

[5]Curiously, we noticed that FaCT is able to classify the knowledge base corresponding

RACER can classify none of the knowledge bases corresponding to the complete UML class diagrams. Instead, it can classify the knowledge bases corresponding to the weakened class diagrams with no multiplicity constraints, and those corresponding to the class diagrams with no minimal multiplicity constraints, with the exception of Restaurant. The weakened class diagrams with no maximal multiplicity constrains are too complex for the current version of RACER, with the exception of Library, where only few minimal multiplicity constraints appear.

From an analysis of the UML class diagrams and the corresponding knowledge bases, it appears that what makes reasoning difficult for the current systems is the combination of: (1) terminological cycles involving existentials (which in UML class diagrams are generated by minimal multiplicity constraints); (2) inverse roles (which are intrinsic in the possibility of navigating UML aggregations and associations components in both directions); (3) functional restrictions combined with existential restrictions (which are present in the complete class diagrams); (4) the overall size of the UML class diagrams.

More information about the conducted experiments, including the $\mathcal{DLR}$ knowledge bases corresponding to the UML class diagrams considered here, and the knowledge bases expressed in the languages accepted by FACT and RACER, are available at `http://www.dis.uniroma1.it/~berardi/uml2dl`.

5 Conclusions

We have seen that UML class diagrams can be formalized as DL knowledge bases, and this potentially allows for exploiting DL-based reasoning systems to perform various kinds of reasoning on them. However, the experimentation with state-of-the-art DL reasoners, shows that the current reasoners may have serious efficiency problems in dealing with the resulting knowledge bases. Observe that all results obtained apply also to Entity-Relationship diagrams (with cardinality constraints) [6, 5], which are tightly related to UML class diagrams.

Hence, we encourage further research on practical DL reasoners. Reasoning with UML class diagrams (with multiplicity constraints) can be a challenging testbed for them.

Acknowledgments We would like to thank the developers of FACT and RACER, and in particular Ian Horrocks, Sergio Tessaris, Volker Haarslev, and

to Restaurant, if we reverse the direction of two aggregations (related and is_comprised), which in this case amounts to reversing the order of the two arguments of the $\mathcal{DLR}$ relation corresponding to two aggregations. This appears quite strange, considering that $\mathcal{DLR}$ relations are reified in $\mathcal{ALCQI}$ and the treatment of the two components in the translation of the relations is completely symmetrical.

Ralf Möller, for their kind and very helpful assistance during the experimentation with their systems.

References

[1] G. Booch, J. Rumbaugh, and I. Jacobson. *The Unified Modeling Language User Guide*. Addison Wesley Publ. Co., Reading, Massachussetts, 1998.

[2] 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.

[3] 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.

[4] D. Calvanese, G. De Giacomo, and M. Lenzerini. Identification constraints and functional dependencies in description logics. In *Proc. of IJCAI 2001*, 2001. To appear.

[5] D. Calvanese, G. De Giacomo, M. Lenzerini, D. Nardi, and R. Rosati. Use of the reconciliation tool at Telecom Italia. Technical Report DWQ-UNIROMA-007, DWQ Consortium, Oct. 1999.

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

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

[8] A. Evans, R. France, K. Lano, and B. Rumpe. The UML as a formal modeling notation. In H. Kilov, B. Rumpe, and I. Simmonds, editors, *Proc. of the OOPSLA'97 Workshop on Object-oriented Behavioral Semantics*, pages 75–81. Technische Universität München, TUM-I9737, 1997.

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

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

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

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

[13] V. Haarslev and R. Möller. RACER system description. In *Proc. of IJCAR 2001*, 2001.

[14] D. Harel and B. Rumpe. Modeling languages: Syntax, semantics and all that stuff. Technical Report MCS00-16, The Weizmann Institute of Science, Rehovot, Israel, 2000.

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

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

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

[18] T. Kirk, A. Y. Levy, Y. Sagiv, and D. Srivastava. The Information Manifold. In *Proceedings of the AAAI 1995 Spring Symp. on Information Gathering from Heterogeneous, Distributed Enviroments*, pages 85–91, 1995.

[19] D. McGuinness and J. Wright. Conceptual modelling for configuration: A description logic-based approach. *Artificial Intelligence for Engineering Design, Analysis, and Manufacturing Journal*, 12:333–344, 1998.

[20] U. Sattler. *Terminological Knowledge Representation Systems in a Process Engineering Application.* PhD thesis, LuFG Theoretical Computer Science, RWTH-Aachen, 1998.

A Appendix

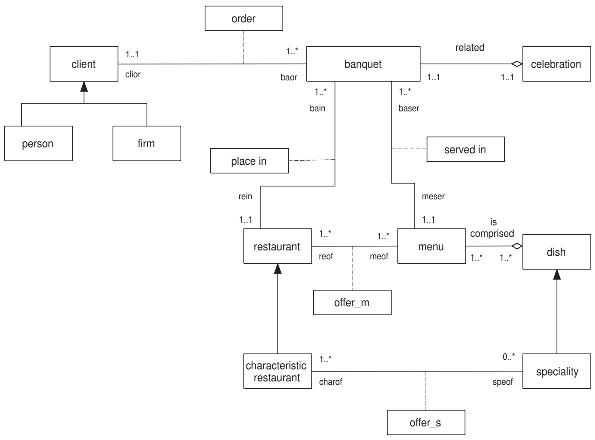

Figure 6: UML class diagram: RESTAURANT

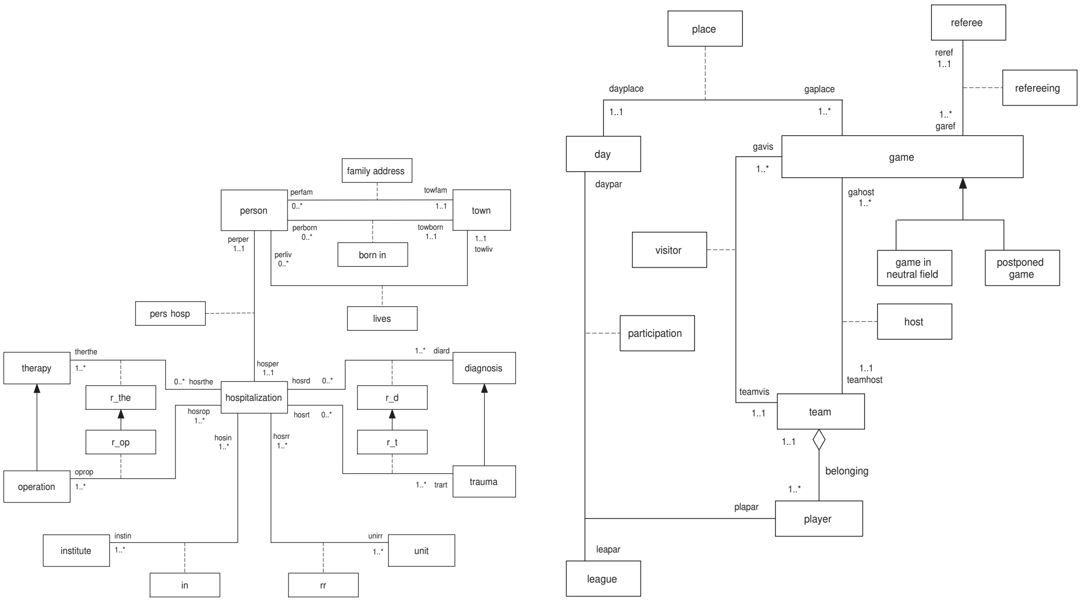

Figure 7: UML class diagrams: HOSPITAL and SOCCER

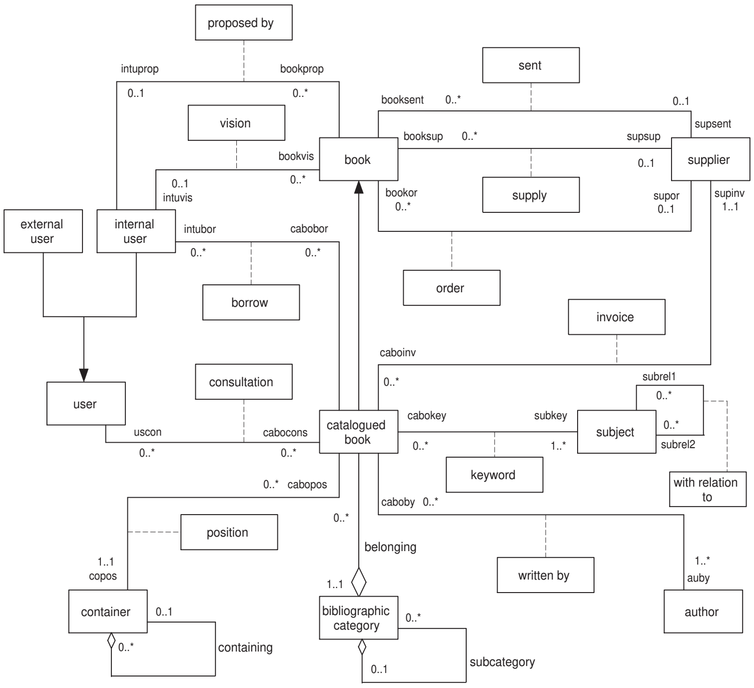

Figure 8: UML class diagram: LIBRARY