Finite Satisfiability of UML class diagrams by Constraint Programming

Toni Mancini

Dipartimento di Informatica e Sistemistica
Università di Roma "La Sapienza"
Via Salaria 113, I-00198 Roma, Italy
tmancini@dis.uniroma1.it
www.dis.uniroma1.it/~tmancini

The Unified Modelling Language (UML, cf. www.uml.org) is probably the most used modelling language in the context of software development, and has been proven to be very effective for the analysis and design phases of the software life cycle.

UML offers a number of diagrams for representing various aspects of the requirements for a software application. Probably the most important diagram is the *class diagram*, which represents all main structural aspects of an application: *classes*, *associations*, *ISA hierarchies* between classes, *multiplicity constraints* on associations. Actually, a UML class diagram represents also other aspects, e.g., the attributes and the operations of a class, the attributes of an association, and the specialization of an association. Such aspects, for the sake of simplicity, will not be considered.

A class diagram induces restrictions on the number of objects, because of its multiplicity constraints. In some cases the number of objects of a class is forced to be zero. When a class is forced to have either zero or infinitely many instances, it is said to be *finitely inconsistent* or *finitely unsatisfiable*.

Unsatisfiability, either finite or unrestricted, of a class is a symptom of a bug in the analysis phase, since such a class is clearly superfluous. In particular, finite unsatisfiability is especially relevant in the context of applications, e.g., databases, in which the number of instances is intrinsically finite.

Finite inconsistency may arise in complex situations, and discovering finite inconsistency may be not doable by hand, since it requires global reasoning on the whole class diagram. Thus, automated finite model reasoning in UML class diagrams (including checking finite satisfiability of classes) is of crucial importance for assessing quality of the analysis phase in software development. Indeed, for the purpose of software engineering, finite model reasoning in UML class diagrams is often considered more important than unrestricted reasoning.

In our research we address the implementation of finite model reasoning on UML class diagrams, a task that has not been attempted so far. This is done by exploiting an encoding of UML class diagrams in terms of Description Logics (DLs).

Reasoning in such logics has been studied extensively, both from a theoretical point of view, establishing EXPTIME-completeness of various DL variants, and from a practical point of view, developing practical reasoning systems. The correspondence between UML class diagrams and DLs allows one to use the current state-of-the-art DL reasoning systems to reason on UML class diagrams. However, the kind of reasoning that such systems support is unrestricted (as in first-order logic), and not finite model reasoning. That is, the fact that models (i.e., instantiations of the UML class diagram) must be finite is not taken into account.

Interestingly, in DLs, finite model reasoning has been studied from a theoretical perspective, and its computational complexity has been characterized for various cases. However, no implementations of such techniques have been attempted till now. In our research we reconsider such work, and on the basis of it we present, to the best of our knowledge, the first implementation of finite model reasoning in UML class diagrams.

The main result of our current reseach is that it is possible to use off-the-shelf tools for constraint modelling and programming for obtaining a finite model reasoner. In particular, we propose an encoding of UML class diagram satisfiability as a Constraint Satisfaction Problem (CSP). Moreover, we show also how constraint programming can be used to actually return a finite model of the UML class diagram.

We built a system that accepts as input a class diagram written in the MOF syntax, and translates it into a file suitable for ILOG's OPLStudio, which checks satisfiability and returns a finite model, if there is one. The system allowed us to test the technique on the industrial knowledge base CIM, obtaining encouraging results.