Finite Satisfiability of Class Diagrams: Practical Occurrence and Scalability of the *FiniteSat* Algorithm

Victor Makarenkov[1], Pavel Jelnov[2], Azzam Maraee[1] and Mira Balaban[1]

[1] Computer Science Department
Ben-Gurion University of the Negev, Beer-Sheva 84105, Israel
makarenk@cs.bgu.ac.il, mari@cs.bgu.ac.il, mira@cs.bgu.ac.il

[2] Department of Mathematics and Statistics
Bar-Ilan University, Israel
pavlikjel@gmail.com

Abstract. Models lie at the heart of the emerging Model Driven Development (MDD) approach, in which software is developed by repeated transformations of models. Since models are intended as executable specifications, there is a need to provide correctness management on the model level. The underlying hypothesis of this research is that model level tools should be strengthened, to support model elements in a way that would encourage users to take advantage of their features. Furthermore, model transformations should not neglect the translation of model features.
This paper explores the practical relevance of detecting Finite Satisfiability problems on the model level. The frequency of occurrence of Finite Satisfiability problems, and the scalability of the efficient ***FiniteSat*** algorithm are studied on a set of synthetic class diagrams, created along designed metrics. The contribution of this work is twofold, first in advancing towards creating a benchmark of class diagrams, and second, in the empirical study of the Finite Satisfiability problem.

Keywords:
UML class diagram, finite satisfiability occurrence, detection, multiplicity constraints, linear programming reduction, scalability, large models, benchmarking, statistical significance.

MoDeVVa'09, October 5, Denver, CO, United States

1 Introduction

Models lie at the heart of the emerging Model Driven Development (MDD) approach, in which software is developed by repeated transformations of models. In this paradigm, models are no longer restricted design artifacts, but play a central role in the process of software development. Therefore, it is essential to have precise, consistent and correct models.

It seems that vendors are not aware that a large portion of designs might have correctness problems of some kind. If models are intended as executable specifications, there is a need to provide correctness management on the model level.

The Unified Modeling Language (UML) is nowadays the widely accepted standard for modeling systems. It consists of a variety of visual modeling diagrams, each describing a different view of object-oriented software. *Class Diagrams* are probably the most important and best understood among all UML models. A class diagram provides a static description of system components. It describes system structure in terms of classes, associations, and constraints.

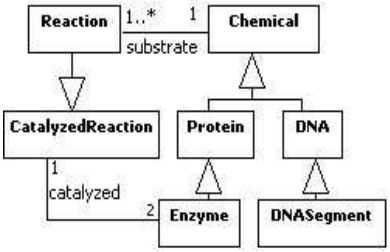

Fig. 1. A Class Diagram with a Finite Satisfiability Problem

Figure 1 is an example of a class diagram, which partially specifies an ontology in the biology domain. It includes association, class hierarchy and multiplicity constraints. The constraints are inter-woven in a complex way. For example, class Enzyme participates in class hierarchy constraints and association constraints with non-trivial multiplicity constraints imposed on it as well.

Class diagrams might pose various correctness problems, that are caused by the interaction of constraints. The major correctness problems are *consistency* and *finite satisfiability*. Consistency involves the ability to instantiate the classes in a diagram without contradicting any constraint and having non-empty classes. Finite satisfiability requires in addition finite population for classes. Both problems are known to be EXPTIME-complete for UML class diagrams [1, 2].
Finite Satisfiability is caused by cyclic interactions involving non trivial multiplicity constraints(different from 0 or *). For example, the diagram in Figure 1 is not finitely satisfiable due to interaction of multiplicity and class hierarchy constraints among the Chemical, Reaction, CatalyzedReaction, Enzyme and Protein

classes. The non-trivial multiplicity constraints in this cycle are 1 and 2). In general, since class diagrams in large systems can be quite big and are written by people, they cannot be expected to be error free.

Computer Aided Software Engineering (CASE) tools do little analysis on the model level (compare with commercial Integrated Development Environments (IDEs) that provide great programming aid on the code specific level). Moreover, many model level constraints are ignored by model-to-code transformations. For class diagrams, almost all visual constraints, including multiplicity constraints, are ignored. Given this situation, it is not surprising that software modelers do not frequently use non trivial multiplicity constraints. Yet, in databases, for example, multiplicity constraints are frequently imposed on tabular relations. In general, non trivial multiplicity constraints appear naturally in real application requirements, such as law systems.

The underlying hypothesis of this research is that model level tools should be strengthened, to support model elements in a way that would encourage users to take advantage of their features. Furthermore, model transformations should not neglect the translation of model features. In this work we concentrate on the problem of detecting Finite Satisfiability problems on the model level. In order to do so, we answer two questions: 1) Does the problem really appear in class diagrams and if it does, how often 2) What is the scalability of the ***FiniteSat*** [3] algorithm (which provides an efficient procedure for detection of Finite Satisfiability problems in a large fragment of the UML Class Diagram model).

In order to answer these questions, there is a need for a large set of class diagrams, that includes non-trivial multiplicity constraints (the cause of Finite Satisfiability problems), and that can be accepted as a representative set. Unfortunately, such a set does not exist, for two reasons: 1) As explained above, users neglect specification of non-trivial multiplicity constraints; 2) There is no accepted benchmark for class diagrams. Consequently, as part of this work, we study and develop a set of metrics for measuring the size and complexity of class diagrams. The frequency of Finite Satisfiability occurrence, and the scalability of ***FiniteSat*** are studied on a set of synthetic class diagrams, created along the designed metrics. The contribution of this work is twofold, first in advancing towards creating a benchmark of class diagrams, and second, in the study of the above two questions concerning the Finite Satisfiability problem. These questions are also checked on several medium size class diagrams of real applications.

Section 2 provides the relevant terminology and describes our earlier work. Sections 3 discusses the issue of class diagram generation, and Section 4 describes the experimental studies of the Finite Satisfiability problem. Section 5 concludes the paper, and points to future research.

2 Background

The subset of UML class diagrams considered in this paper includes *classes* with attributes, *associations* and five kinds of constraints:

1. *Multiplicity (cardinality)* constraints on binary associations.
2. *Class hierarchy* constraints.
3. *Generalization set* (*GS*) constraints.
4. *Qualifier* constraints, that strengthen multiplicity constraints.
5. *Association class* constraints , that reify associations.

The standard set theoretic semantics of class diagrams associates a class diagram with *instances* in which class extensions are sets of objects and association extensions are relationships among class extensions. Detailed semantics is described in [3].

A *legal instance* of a class diagram is an instance that satisfies all constraints. Correctness of a class diagram involves *consistency* and *finite-satisfiability* [4, 5, 6, 1, 7].

A *class is consistent* if it has a non-empty extension in some legal instance; a *class diagram is consistent* if all of its classes are consistent; a *class is finitely satisfiable* if it has a non-empty extension in some legal finite instance; a *class diagram is finitely satisfiable* if all of its classes are finitely satisfiable[1].

2.1 Finite Satisfiability of UML Class Diagrams

There are two main approaches for reasoning about finite satisfiability of class diagrams: The *linear inequalities approach* and the *graph based approach*. The first approach reduces the finite satisfiability problem into solvability of a linear inequality system. The second approach identifies infinity causing cycles of multiplicity constraints in the diagram. All methods apply only to fragments of UML class diagrams. Deciding infinity in unrestricted class diagrams is still an open issue.

The fundamental work in the linear inequalities approach is that of [5]. It applies to *Entity-Relationship* (*ER*) diagrams with *Entity Types* (*Classes*), *Binary Relationships*[2] (*Associations*), and *multiplicity Constraints*. The method transforms the multiplicity constraints into a system of linear inequalities whose size is polynomial in the size of the diagram.

Calvanese and Lenzerini, in [4], extend the inequalities based method of [5] to apply to diagrams with class hierarchy constraints, but the size of the resulting system of inequalities is exponential in the size of the class diagram. ***Finite-Sat*** [3] is a polynomial time algorithm, that extends the method of [5] for the following constraints: multiplicity, class hierarchy, generalization set, qualifier and association class. The scope of the algorithm is restricted by interaction of

[1] Lenzerini and Nobili [5] used the notion of *strong satisfiability* for this term.

[2] They allow also n-ary relationships, but with non-standard (membership) semantics for cardinality constraints.

disjoint, complete Generalization Set constraints with multiple inheritance structures. Applying ***FiniteSat*** to the class diagram in Figure 1, the algorithm produces the inequality system below. The symbols c, r, cr, e, p, d, and ds stand for the classes $Chemical$, $Reaction$, $CatalyzedReaction$, $Enzyme$, $Protein$, DNA and $DNASegment$, respectively. The symbols s, t, and i stand for the associations *substrate*, *catalyzed* and *comprise*, respectively.

1. Class hierarchy and multiplicity constraints among classes $Chemical$, $Protein$, $Enzyme$, $Reaction$ and $CatalyzedReaction$: $s = r,\ s \geq c,\ t = 2cr,\ t = e,\ cr \geq r,\ p \geq e,\ c \geq p$.
2. Class hierarchy and multiplicity constraints among classes $Chemical$, DNA and $DNASegment$:$c \geq d,\ i \geq d,\ i = ds$.
Non-emptiness of classes and associations: $c > 0,\ r > 0,\ cr > 0,\ e > 0,\ p > 0,\ d > 0,\ ds > 0,\ s > 0,\ t > 0,\ i > 0$.

The inequalities in the class cycle (1) imply $c \geq 2c$, and together with the non-emptiness inequalities, the inequality system is unsolvable. The correctness result for the ***FiniteSat*** algorithm implies that the class diagram in Figure1 is not finitely satisfiable.

A graph based method for identification of the cause for non finite satisfiability was first suggested in [5]. [3] extends this method for class hierarchy, qualifier and association class constraints.

3 Towards a Benchmark for Class Diagrams

Creating a benchmark for class diagrams requires a set of metrics for measuring features of class diagrams. Such metrics should direct the selection or creation of benchmark members. Conventional metrics for software include, for example, "number of lines", "number of fields" and "number of methods" in object-oriented programming [8]. For UML class diagrams, [9, 10] suggest the following metrics, for measuring the cognitive complexity of class diagrams:

1. NCM - Number of Classes in a Model
2. $NASM$ - Number of ASsociations in a Model
3. $NSUBC$ - Number of SUBclasses of a Class
4. $NSUPC$ - Number of SUPerclasses of a Class
5. DIT - Depth of Inheritance Tree

We suggest a distinction between *size* and *structure* metrics. Obvious size metrics for class diagrams are the NCM and $NASM$ above. Structure metrics measure constraints in a class diagram, and their inter-relations (or ratio). The last three metrics above can be considered as structure metrics. Additional structure metrics might be:

1. $\frac{NASM}{NCM}$ - Ratio between number of associations and number of classes.
2. $NoTM$ - Number of non-Trivial Multiplicity constraints.

3. $NCYC$ - Number of cycles formed by constrained associations and classes.
4. $\frac{NoTM}{NCM}$ or $\frac{NoTM}{NASM}$ - The ratio between the number of non-trivial multiplicity constraints and the number of classes or of associations. This metric assumes that every association is constrained by a trivial $0..*$ multiplicity constraint.
5. $\frac{NCH}{NCM}$ - The ratio between of number of class hierarchy constraints to the number of classes.
6. $\frac{NGS}{NCH}$ - The ratio between the number of generalization set constraints and the number of class hierarchy constraints.
7. $\frac{NMCH}{NCH}$ - The ratio between the number of multiple inheritance constraints to the number of class hierarchy constraints. This metric seems especially relevant for experiments using the ***FiniteSat*** algorithm, since interaction between such structures and certain generalization set constraints determine the scope of this algorithm [7].

Size and structure metrics are both essential: They complement each other. The size metrics are necessary for having large (or small) size class diagrams, while the structure metrics are necessary for measuring the structural complexity of diagrams.

A benchmark of class diagrams of real applications, that actually use a meaningful portion of the Class Diagram model is not feasible, since modelers still do not use many features of this model language. As noted above, this might be a chicken-and-egg phenomenon of modelers that do not use features not supported by tools, and tools that do not support features that modelers do not use. Therefore currently, only a synthetic benchmark is feasible.

We have implemented a platform for generation of synthetic class diagrams, based on selection of size and structure metrics. The generation follows the metaphore of a random graph model of Erdos and Renyi [11]. In this model, undirected edges are placed at random between a fixed number n of vertices to create a graph in which each of the $\frac{1}{2}n(n-1)$ posssible edges is independently present with some probability p. In the class diagram paradigm, *Classes* play the role of *vertices*, and *associations* and constraints play the role of *edges*. The probability p is determined by the constraint metric. For example, for the metric $\frac{NoTM}{NCM}$ that specifies the ratio between non-trivial multiplicity constraints to number of classes, the ratio determines the probability. Note that this metaphor allows duplicate edges.

The implemented platform uses a class diagram symbolic representation that relies on USE [12]. The platform has a process application unit that applies an algorithm to the symbolic representation. For the Finite Satisfiability problem testing, it applies an implementation of the ***FiniteSat*** algorithm, that uses an off-the-shelf linear programming component.

4 Finite Satisfiability: Problem Occurrence and Scalability of *FiniteSat*

This section describes the double goal experiments for observing the frequency of Finite Satisfiability problems in class diagrams, and checking the scalability

of the ***FiniteSat*** algorithm. The experiments described in this paper, use the NCM and the $\frac{NoTM}{NCM}$ metrics. The reason is that Finite Satisfiability is affected only by non-trivial multiplicity constraints (therefore, the number of associations in general has no impact on the tested question). Note that the size metric NCM cannot be neglected as it provides an essential measure for both, frequency and scalability.

4.1 First Experiment: Occurrence of the Finite Satisfiability problem, and *FiniteSat* scalability

In this experiment, class diagrams are repeatedly generated, and ***FiniteSat*** is applied. Table 1 demonstrates the results of the experiment, for very large class diagrams (100 to 500 classes). For every metrics combination, the experiment includes 100 and 1000 repetitions, in order to demonstrate high probability results. The first two columns of the table describe the values of the metrics NCM and $\frac{NoTM}{NCM}$ (the ratio is not explicitly stated). The third and fourth columns state the percentage of diagrams in which a Finite Satisfiability problem was detected. The last column describes the average running time over 100 repetitions. The results show very high performance, where class diagrams with 500 classes and 500 non-trivial multiplicity constraints, can be tested by ***FiniteSat*** in 46 seconds. All other results demonstrate less than one minute running times.

Table 1. Existence and Scalability Experiments Results

		Occurrence Experiment		Scalability
NCM	NoTM	N=100	N=1000	Running Time in mili-seconds
50	10	8%	12.1%	7
50	25	34%	35.1%	12
50	50	91%	87.1%	37
100	50	29%	25.9%	112
100	100	96%	89.2%	376
200	100	31%	27%	899
200	200	100%	96.4%	2963
500	100	12%	6.8%	4134
500	250	24%	27.4%	13514
500	500	100%	99.5%	46229

Statistical Analysis of the Experiments Results:
The main question considering the experiment results is whether they are statistically significant. For that purpose, we analyzed the experiment results using the *Z-test* for proportion. The analysis denotes by p_0, a slightly downward rounded percentage of occurrence of Finite Satisfiability problems (compared with the results obtained in the experiments).

For each metric combination, the analysis observes two hypotheses:

Table 2. P-values of one-sided Z-test

NCM	NoTM	p_0	p-value for N=100	p-value for N=1000
50	10	5%	8.43%	**0.00**%
50	25	30%	19.14%	**0.29**%
50	50	85%	**4.64**%	**0.00**%
100	50	25%	17.78%	**0.17**%
100	100	90%	**2.28**%	**0.00**%
200	100	25%	8.29%	**0.00**%
200	200	95%	**1.09**%	**0.00**%
500	100	10%	25.25%	**1.75**%
500	250	20%	15.87%	**0.08**%
500	500	95%	**1.09**%	**0.00**%

1. $H_0 : p = p_0$: The probability of occurrence of a Finite Satisfiability problem is equal to p_0, a positive number which grows as the metrics values grow.
2. $H_1 : p > p_0$: The probability of occurrence of a Finite Satisfiability problem is above p_0.

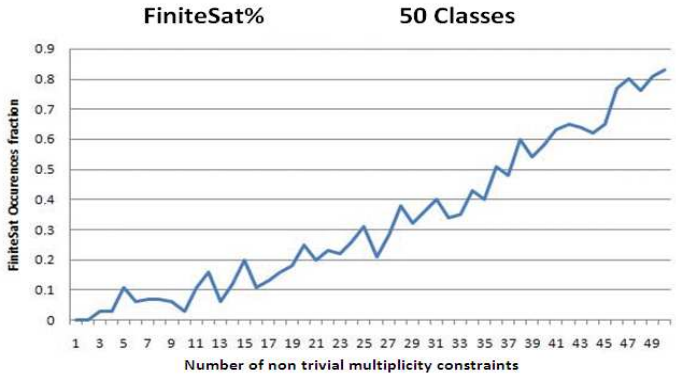

Fig. 2. A Graph showing the appearance of Finite Satisfiability problem, by structure of class diagram. The number of classes is 50.

Assuming that the repetitions in each experiment are independent of each other, and the result of each repetition is 0 or 1 (whether the problem exists or not), we have a binomial distribution for each combination of the metrics' values. According to the Central Limit Theorem we can use the one-sided Z-test for proportion to test our hypotheses. The variable *p-value* denotes the probability to achieve the results of the experiment, or more extreme results, under H_0. The 5% confidence level is often used by statisticians. P-values below 5% show confidence that H_0 is statistically wrong and H_1 is preferable. The p-values of the test are given in table 2.

Meaningful p-values are those below 5% (bolded in Table 2). These results show that for 1000 repetitions, all experiment results are confidently above the proposed p_0. For example, for 50 classes and 10 non-trivial multiplicity constraints, the probability of occurrence of a Finite Satisfiability problem is confidently above 5%; for 50 classes and 25 non-trivial multiplicity constraints, the probability of occurrence of a Finite Satisfiability problem is confidently above 30%; for 500 classes and 100 non-trivial multiplicity constraints, the probability of occurrence of a Finite Satisfiability problem is confidently above 10%. We see that the occurrence of Finite Satisfiability problems is affected both by the size and structure metrics of class diagrams.

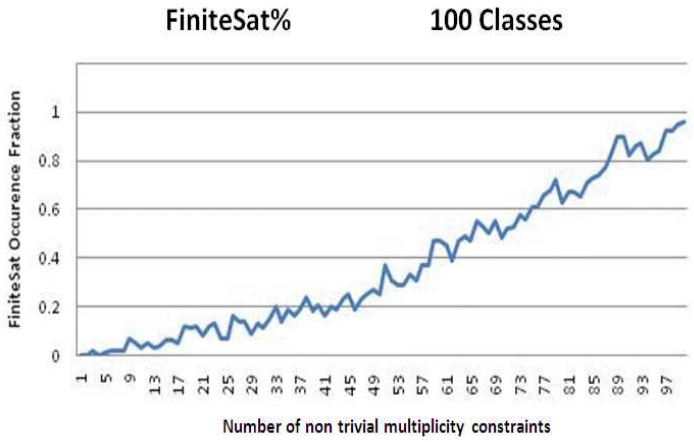

Fig. 3. A Graph showing the appearance of Finite Satisfiability problem, by structure of class diagram. The number of classes is 100.

4.2 Second Experiment: Finite Satisfiability Occurrence

This experiment checks the impact of the $\frac{NoTM}{NCM}$ structure metric on the problem of Finite Satisfiability occurrence. For that purpose, we performed a series of experiments, where the number of classes is fixed, and the number of non-trivial multiplicity constraints grows. Figures 2 and 3 show the percentage of occurrence of Finite Satisfiability problems, when the number of classes is fixed as 100 and as 50, respectively. As expected, greater $\frac{NoTM}{NCM}$ values yield greater occurrence of Finite Satisfiability problems.

5 Future Work

This paper presents an empirical study of practical issues of the Finite Satisfiability problem: The frequency of its occurrence, and the scalability of the

efficient ***FiniteSat*** algorithm. The results show, with high statistical confidence, that the Finite Satisfiability problem occurs frequently, and therefore, cannot be neglected. Furthermore, we received very good scalability results for ***FiniteSat*** . The empirical study also required investigation of metrics for measuring class diagram size and structural complexity.Future work of this research will concentrate on the issue of providing metrics and creating a benchmark of synthetic class diagrams, and on developing incremental efficient methods for implementing the ***FiniteSat*** algorithm. Such methods are, particularly relevant, for distributed applications, where the complete large class diagram is not available.

References

[1] Berardi, D., Calvanese, D., Giacomo, D.: Reasoning on uml class diagrams. Artificial Intelligence **168** (2005) 70–118

[2] Lutz, C., Sattler, U., Tendera, L.: The complexity of finite model reasoning in description logics. Inf. Comput. **199** (2005) 132–171

[3] Maraee, A., Makarenkov, V., Balaban, M.: Efficient recognition and detection of finite satisfiability problems in uml class diagrams: Handling constrained generalization sets, qualifiers and association class constraints. In: 1st International Workshop on "Model co-evolution and consistency management" (MoDELS'08). (2008)

[4] Calvanese, D., Lenzerini, M.: On the interaction between isa and cardinality constraints. In: The 10th IEEE Int. Conf. on Data Engineering, Washington, DC, USA, IEEE Computer Society (1994) 204–213

[5] Lenzerini, M., Nobili, P.: On the satisfiability of dependency constraints in entity-relationship schemata. Information Systems **15(4)** (1990) 453–461

[6] Thalheim, B.: Entity Relationship Modeling, Foundation of Database Technology. Springer-Verlag (2000)

[7] Maraee, A., Balaban, M.: Efficient reasoning about finite satisfiability of uml class diagrams with constrained generalization sets. In: The 3rd European Conference on Model-Driven Architecture, Springer (2007) 17–31

[8] Chidamber, Kemerer: A metrics suite for object oriented design. IEEE Transactions on Software Engineering **20** (1994) 476 – 493

[9] Kim, H., Boldyreff, C.: Developing software metrics applicable to uml models. In: 6th ECOOP Workshop on Quantitative Approaches in Object-Oriented Software Engineering. (2002)

[10] Manso, M.E., Genero, M., Piattini, M.: No-redundant metrics for uml class diagram structural complexity. In: Lecture Notes in Computer Science : Advanced Information Systems Engineering. (2003)

[11] Erdos, P., Renyi, A.: On random graphs. In: Publ. Math. Debrecen. Volume 6. (1959) 290–297

[12] Gogolla, M., Buttner, F., Richters, M.: Use: A uml-based specification environment for validating uml and ocl. Science of Computer Programming (2007) 69:27–34