Management of Correctness Problems in UML Class Diagrams Towards a Pattern-Based Approach

Mira Balaban, Ben-Gurion University of the Negev, Israel
Azzam Maraee, Ben-Gurion University of the Negev, Israel
Arnon Sturm, Ben-Gurion University of the Negev, Israel

ABSTRACT

UML is now widely accepted as the standard modeling language for software construction. The Class Diagram is its core view, having well formed semantics and providing the backbone for any modeling effort. Class diagrams are widely used for purposes such as software specification, database and ontology engineering, meta-modeling, and model transformation. The central role played by class diagrams emphasizes the need for strengthening UML modeling tools with features such as recognition of erroneous models and the detection of errors' sources. Correctness of UML class diagrams refers to the capability of a diagram to denote a finite but not empty reality. This is a natural, unquestionable requirement. Nevertheless, incorrect diagrams are often designed, due to the interaction of contradicting constraints and the limitations of current tools. In this paper, the authors clarify the notion of class diagram correctness, discuss various approaches for detecting correctness problems, and propose a pattern-based approach for identifying situations in which correctness problems occur, and for providing explanations and repair advice.

Keywords: Consistency, Finite Satisfiability, Incorrectness Patterns, Model Driven Engineering, Modeling Languages, Redundancy, UML Class Diagrams

INTRODUCTION

The Unified Modeling Language (UML) is the de facto standard for system development, as it was developed and adopted by the Object Management Group (OMG-UML, 2009). It consists of several diagrammatic languages, each describing a different view of object-oriented software; a system model consists of a collection of such diagrams. The most important view of UML is the static/structural specification which describes a structural abstraction of the real world. This view is expressed by class diagrams, which consist of classes and their descriptors, associations among them, and constraints imposed on both classes and associations. Among the nine visual UML
models, class diagrams appear to be the most clear, intuitive and well defined.

Dobing and Parsons (2006) found that the Class Diagram view is the most frequently used (73%) in their examination of the usage of UML. It was found to be useful in clarifying technical understanding, and for maintaining software documentation. The major usage of UML class diagrams is to specify, visualize, and document systems’ static view. They also serve as a basis for generating implementation artifacts such as code skeleton (Martin, 2006) and database schemata (Blaha et al., 1994), as a means for knowledge representation such as specifying ontologies (Cranefield, 2001; Falkovych et al., 2003; Gasevic et al., 2004; Kabilan & Johannesson, 2004; Kogut, 2002; Timm & Gannod, 2005), and for defining metamodels of other programming, modeling, and specification languages.

Class diagrams are models written by people, and therefore, usually suffer from modeling problems like inconsistency, redundancy, and abstraction errors. Inexperienced designers tend to create erroneous models, but even experienced ones cannot anticipate the implication of a change on an overall model (Sunye et al., 2001). Indeed, Lange et al. (2006) show that model defects often remain undetected, even if practitioners check the model attentively. These problems are empowered when a model originates from different resources. Combined sources are usually overlapping, and the integration yields redundant inconsistent models (Ackermann & Turowski, 2006; Huzar et al., 2004). It is a common belief that such problems can best be solved at the level of models (Jackson & Rinard, 2004).

In view of the wide spread usage of UML class diagrams and the difficulties of producing high quality models, it is essential to equip UML CASE tools with reasoning capabilities for identifying problems within models (Berardi et al., 2005; Cadoli et al., 2004; Hartman, 2001; Jackson & Rinard, 2004). These capabilities can help in detecting design problems, identifying the reasons for these errors, suggesting possible solutions, and providing advice for design improvements (Unhelkar, 2005). The quality of models is especially important for the emerging Model Driven Engineering (MDE) approach, in which software is developed by repeated transformations of models (Stahl et al., 2006).

The major correctness features of class diagrams involve consistency and finite satisfiability. They guarantee the natural, unquestionable requirement, that a diagram can denote a finite but non empty reality: Consistency accounts for non-emptiness, and finite satisfiability accounts for finiteness. Emptiness is caused by contradicting constraints, such as designing a subclass of two necessarily disjoint classes. Non-finiteness is caused by interaction among multiplicity (cardinality) constraints, which restrict the number of interactions between objects of related classes. Contradicting multiplicity constraints, or a contradiction between multiplicity constraints to other constraints, like generalization and association class constraints, impose class multiplicity requirements that can be satisfied only by empty or infinite classes. For example, the class diagram in Figure 1 is not finitely satisfiable, since in every legal instance of this diagram the sets denoted by the classes CatalyzedReaction, Enzyme, Protein, Molecule, Chemical, and GeneralReaction, are either all empty or all infinite.

In this paper we define the consistency and finite satisfiability problems in class diagrams, describe current approaches for detection of these problems and identification of their cause, and suggest a pattern-based approach for creating explanations and repair advices. The paper opens with a presentation of the class diagram model, its semantics and correctness problems, which is followed by a survey of existing methods for management the correctness of class diagrams. The core of the paper is devoted to a presentation of the pattern-based approach for detecting correctness problems within class diagrams, and for providing explanations and repair advices. The summary section draws the line for future research.
Related Content

[www.igi-global.com/article/javaspi-framework-security-protocol-implementation/61152?camid=4v1a](www.igi-global.com/article/javaspi-framework-security-protocol-implementation/61152?camid=4v1a)

An Invariant-Based Approach for Detecting Attacks Against Data in Web Applications
[www.igi-global.com/article/an-invariant-based-approach-for-detecting-attacks-against-data-in-web-applications/109579?camid=4v1a](www.igi-global.com/article/an-invariant-based-approach-for-detecting-attacks-against-data-in-web-applications/109579?camid=4v1a)
Agile Practices in Project Management
www.igi-global.com/chapter/agile-practices-project-management/29515?camid=4v1a

Demand-driven Development of Service Compositions in Organizational Networks
www.igi-global.com/article/demand-driven-development-service-compositions/48517?camid=4v1a