# Chapter 4.11
# Formal Verification of a Subset of UML Diagrams:
## An Approach Using Maude

**Allaoua Chaoui**
*University Mentouri Constantine, Algeria*

**Okba Tibermacine**
*University of Batna, Algeria*

**Amer R. Zerek**
*Engineering Academy, Libya*

## ABSTRACT

*We introduce an approach that deals with the verification of UML collaboration and sequence diagrams in respect to the objects internal behaviors which are commonly represented by state machine diagrams. The approach is based on the translation of theses diagrams to Maude specifications. In fact, Maude is a declarative programming language, an executable formal specification language, and also a formal verification system, which permit the achievement of the approach goals. We define in details the rules of translating UML diagrams elements into their corresponding Maude specifications. We present the algebraic structures that represent the OR-States and the AND-states in a state machine diagram, and the structure that represents the collaboration and the sequence diagrams. Also, we explain the mechanism of the execution and the verification of the translated specification, which is based on rewriting logics rules.*

## INTRODUCTION

The Unified Modeling Language (UML) (Rumbaugh, 1999) is widely used language for the specification of object - oriented software systems, including concurrent and embedded systems. An UML model is a set of diagrams describing and documenting the structure, behavior and the usage of a software system. The UML case tools available in today markets help designers to create models and generate code automatically from specific diagrams. Nevertheless, the most of these tools do not offer methods for the verification neither for the validation of these established diagrams,

DOI: 10.4018/978-1-61350-456-7.ch4.11

and this is due to the semantics of UML, which are sometimes inadequate in respect to the desired behaviors.

The need of formal semantics was already discussed by (France, 1998). Also, it's recognized that formal, unambiguous, yet readable account of UML semantics would be very beneficial for the language, the model verification, and in general the oriented object software development. Hence, a lot of emerged semantics approaches attended to formalize the unified notation. They focalized on the state machine diagram. Some of these approaches are purely mathematical models; some are rewriting based systems, and some are translating approaches (Crane, 2005). Generally, the translating approaches are based on the transformation of the UML models into formal pieces ready to be verified by model-checking tools. Model checking (Clarke, 1999) is well-studied technique of automatic formal verification that ensures correctness of a given specification. In literature, some approaches like (Knapp, 2002), (Latella, 1999) and (Lilius, 1999) rely on translating UML Models into languages of model-checking to analyze and verify them. The disadvantage of these approaches is that the semantics model and the verification model aren't the same, and that due to the fact that some model-checking languages like PROMELA/SPIN or SMV are not truly formal languages (Compton, 2000) (Shen, 2002).

In this work, we propose an approach to Verify UML collaboration diagrams against the behavior represented by state machines. The verification is performed after translating the UML model to a formal rewriting logic specification within the *Maude* language. Maude supports declarative programming and executable formal specifications. Inductive theorem proving, model-checking and other formal analysis are either supported by Maude and its formal environment (Meseguer, 2002).

The rest of this chapter is organized as follows; in section 2, we recall some basic definitions of UML classes, state machine and collaboration diagrams. In section 3, we present rewriting logic and Maude language. In section 4, we present the verification of UML using Maude. In section 5, we discuss features of translation and UML models elements. In section 6, we talk about the verification of the specification and the last section concludes the work and gives some perspectives.

## UML CLASS, STATE MACHINE, AND COLLABORATION DIAGRAMS

The basic element for modeling oriented object systems is the active object. An active object has its own thread of control and runs in concurrency with other active objects. A UML class diagram may represent classes of active objects and associations between them.

In this paper we use a simple model of an Automatic Teller Machine (ATM), as it's represented in (Knapp, 2002). Figure 1 shows a class diagram that specifies two active classes ATM and Bank. The association between the two classes rely an instance of Bank to an instance of ATM, and vice versa. Classes define attributes, operations and

*Figure 1. Class diagram*