Chapter VIII

Formalizing UML Class Diagrams

Ana María Funes
Universidad Nacional de San Luis, Argentina

Chris George
UNU/IIST, Macau

ABSTRACT
Graphical notations have demonstrated usefulness when interacting with end users, making system validation easier. However, while they can be easily communicated, they lack precision, which consequently can be ambiguous and lead to misunderstanding. It is here where formal specifications can play an important role in overcoming this drawback. In this chapter, we use the RAISE Specification Language (RSL), which is the language of the formal method RAISE (i.e., Rigorous Approach to Industrial Software), as a syntactic and semantic reference for UML. We present the semantics for UML class diagrams by using RSL as a formal basis. An automated tool that implements the translation and the abstract syntax in RSL for the RSL-translatable class diagrams are also presented.

INTRODUCTION
A common practice among software engineers is the use of diagrams for building their system models. Graphical notations become useful for interacting with the end users. They can be more intuitive and easier to grasp than textual descriptions as, according to folk wisdom, a picture is worth a thousand words.

Different forms of more or less formal diagrams have been used in software engineering for a long time. Many practitioners have adopted the use of graphical notations such as data flow diagrams (DeMarco, 1979), entity-relationship diagrams (Chen 1976), state charts (Harel, 1987) and process-oriented approaches like Jackson System Development (Jackson, 1982; Sutcliffe, 1988). Lately, object-oriented approaches
seem to be the most popular. Object orientation has evolved from a programming paradigm to methods that cover the complete life cycle. There are many variations of these methods such as OMT, i.e., Object Modeling Technique, (Rumbaugh et al., 1991); OOSE (Object-Oriented Software Engineering) (Jacobson et al., 1992), Booch methods (Booch, 1991; Booch 1994), and the Unified Software Development Process (Jacobson et al., 1999). The latter has unified the notation used by the previous methodologies through the use of the Unified Modeling Language (UML) (Booch et al., 1999).

Today, UML has become the de facto standard for object-oriented modeling, and there are a wide variety of graphical tools that support it. UML is a graphical language for modeling and specifying software systems. It consists of a set of constructs common to most object-oriented languages. However, while UML notations are easily communicated, their semantics are informal, which can consequently make them ambiguous, leading to misunderstandings. Here, formal specifications can play an important role as they can be used as formal foundations for expressing the semantics of the semi-formal language. By formal specifications, we refer to the representation of software by means of formal notations (i.e., a language with a precise syntax, precise semantics, and a proof system). The use of formal specifications has benefits ranging from the possibility of building unambiguous specifications, to the possibility of proving system properties, to automatic code generation. As a result, they are an important technique for increasing the reliability of software. However, it is not used frequently in the industry as it requires a high level of expertise in algebra and mathematical logic, making communication with the end users for validating requirements difficult.

As a consequence, a combination of the advantages of both approaches has been proposed; namely, that the use of graphical and formal notations can be achieved in order to overcome the inherent problems without loss of the respective benefits, i.e., the understandability of graphical notations and the lack of ambiguity of formal specifications. Another important consequence of having a formal representation of an informal or semi-formal model is the possibility of reasoning about their properties. Since models are the result of a creative activity, they tend to contain errors and inconsistencies. This makes model verification a very important activity.

There are several studies in the literature using formal techniques for expressing the semantics of object-oriented models such as those carried out by France (1999), DeLoach and Hartrum (2000), Meyer and Souquieres (1999), and Kim and Carrington (2000). In this work, we explore the use of RSL (George et al., 1992), which is the language of the RAISE development method (George et al., 1995), to give the formal foundations for UML class diagrams. We give the semantics for UML class diagrams in RSL. An automated tool that implements the translation and the abstract syntax in RSL for the RSL-translatable class diagrams is also presented.

The general structure of the chapter is as follows: Section “Class Diagrams and UML” introduces class diagrams in UML and their use in different stages of the software development process. The next section introduces the concepts of semantic and syntactic reference and gives a concise description of RSL and the RAISE method. In the section “Formal Syntax”, the abstract syntax in RSL for the considered subset of UML class diagrams is given. Section “Formal Semantics” presents the formal semantics in RSL for class diagrams. Section “RSL Templates” contains a set of RSL templates resulting from the analysis carried out when giving the semantics of UML class diagrams.
Related Content

Using a Graph Transformation System to Improve the Quality of Characteristics of UML-RT Specifications
www.irma-international.org/chapter/using-graph-transformation-system-improve/4929/

Version Control of Software Models
www.irma-international.org/chapter/version-control-software-models/4930/

Content-Based Publish/Subscribe for XML Data
www.irma-international.org/chapter/content-based-publish-subscribe-xml/41506/

Graphical Notations for Rule Modeling
www.irma-international.org/chapter/graphical-notations-rule-modeling/35855/

PRAISE: A Software Development Environment to Support Software Evolution
www.irma-international.org/chapter/praise-software-development-environment-support/4933/