Chapter VIII

Is There a Difference Between the Theoretical and Practical Complexity of UML?

Keng Siau, University of Nebraska – Lincoln, USA
John Erickson, University of Nebraska at Omaha, USA
Lih Yunn Lee, University of Nebraska – Lincoln, USA

Abstract

An on-going and major problem faced by information systems developers and business users alike is reaching a clear and consensual understanding of the system by both groups. This can be difficult because the businesses are (typically) process driven, while the systems are (increasingly) object-oriented. Enter modeling. Modeling is one way of presenting complex information in a way that enhances or eases understanding. But, even models can be extremely complex, and the underlying tools and modeling languages are not any less complex. This chapter investigates the possibility that modeling languages can be simplified by considering that not all of the “words” in the language are used all of the time. If theoretical (maximum) represents all the words in a modeling language, then this chapter suggests that there might exist a more use-based (we name it practical complexity) subset that represents an easier to learn and use subcomponent of the language.
Introduction

The move toward object-oriented (OO) programming languages by application developers, in the past decade, has addressed a number of important issues, but, simultaneously, presented other problems to systems developers. One of the top issues confronting developers is that the systems resulting from the shift are quite naturally object-oriented systems. While that may not sound like a big problem at first glance, the fact that much of the systems analysis and design is still being conducted, at least partially, from a structured or more “traditional” perspective. At best, this leaves the (typically business) clients in a bit of a state of cognitive dissonance, and, to a certain extent, developers and programmers also, when it comes to understanding and explaining the systems.

The differences between the structured and OO methodologies are quite pronounced in some ways. As a response to these problems and issues, object-oriented systems analysis and design gradually emerged from object-oriented programming. Not that this made the situation entirely better, since there have been more than 50 OO methodologies proposed and used since 1990. However, over time, three of those modeling methodologies were combined and merged to form a new analysis and design technique called the unified modeling language (UML) (Booch et al., 1999). UML 1.5 consisted of nine distinct diagramming techniques designed to support object-oriented systems development, while UML 2.0 has increased the number of diagramming techniques to 13. The Object Management Group (OMG) adopted the UML language as a standard in 1997. Since that time, a growing number of researchers and practitioners have found UML to be extremely heavy and complex, both to learn and to use. In addition, and perhaps just as importantly, many of the UML constructs have been found to be ambiguous as to meaning and clarity. (Siau, Erickson, & Lee, 2002; Siau & Loo, 2002; Siau & Lee, 2005). Support for extensions to allow UML to be used in a variety of domains was also extremely lacking in versions 1.X, but this concern has begun to be addressed in version 2.0.

Siau and Cao (2001) used the complexity metrics developed by Rossi and Brinkkemper (1996) to analyze the nine diagramming techniques in UML and compared them to other modeling methods. Here, we offer the idea that the complexity metrics developed by Rossi and Brinkkemper (1996) and used to analyze UML (Siau & Cao, 2001) can be seen to do two tasks with regards to understanding and quantifying the complexity of UML. First, this approach identifies one level of complexity, what could be considered as the theoretical or maximum complexity of the UML modeling techniques. This level of complexity is composed using all of the constructs from the meta-model, as proposed by Rossi and Brinkkemper (1996). The term meta-model refers to a model of the model. For example, in the case of UML, it means a description of all possible constructs in each diagramming technique, in terms of three components: object types, relationship types, and property types.
Dynamic Semantics of UML State Machines: A Metamodelling Perspective
www.igi-global.com/article/dynamic-semantics-uml-state-machines/3285?camid=4v1a

Database Design Based on B
www.igi-global.com/chapter/database-design-based/7925?camid=4v1a