Available online at www.sciencedirect.com
SCIENCE DIRECT®
Electronic Notes in Theoretical Computer Science
ELSEVIER Electronic Notes in Theoretical Computer Science 130 (2005) 57–77
www.elsevier.com/locate/entcs

Towards a Rigorous Approach to UML-Based Development [1]

Zhiming Liu and He Jifeng

International Institute for Software Technology
The United Nations University, Macao SAR, China

Xiaoshan Li

Faculty of Science and Technology, University of Macau, Macao SAR, China

Abstract

We discuss the promises and problems of UML-based development. We then suggest a framework in which UML can be used precisely and more disciplined so as to solve the problems and meet the promises better.

Keywords: Object-Orientation, Component-Based Development, Refinement, Specification, Transformation

1 Introduction

When programming in the small, the *specification* of the problem is mainly concerned with the control and data structures of the program. The program development is the design and implementation of data structures and algorithms through a number of steps of *refinement*. *Verification* is needed to prove that each step preserves the specification of the control and data structures in the previous step. Various formal methods, especially those state-based

[1] Email: lzm@iist.unu.edu,hjf@iist.unu.edu,xsl@umac.mo.
J. He is on leave from East China Normal University, Shanghai, China.
J. He's work is partly supported by the research grant 02104 MoE and the 973 project 2002CB312000 of MoST of P.R. China.

doi:10.1016/j.entcs.2005.03.005

models [9,18] such as VDM [20] and Z [8], are widely found helpful in correct and reliable construction of such a program.

For programming in the large, problems become more complicated. The specification has to be described in terms of (or decomposed into) *components*, their *interfaces* and *interactions*. Such a specification in general contains different views and aspects, e.g. the static view, the interaction view and the behavioural view. An overall structure, i.e. the system architecture, is required to *consistently* unify these views. The system construction needs a process of model transformations. These transforms have to ensure consistency of the deferent views and preserve the functional and behavioural correctness. Therefore, for programming in the large, it is ideal in general to have a multi-view modelling language that supports specifications at different levels of abstraction. In this article, we will discuss how UML can be used for this purpose and how it can be used formally for correct and reliable construction of large scale software.

1.1 The promises of UML

UML2.0 is designed as a modelling language for *component-based* and *object-oriented* development, promising to support programming in the large. It is obviously a multi-view and multi-notional language and we can count up to at least 10 kinds of diagrams including component diagrams, packages, class diagrams, object diagrams and use-case diagrams for static views; activity diagrams, interaction diagrams (sequence diagrams and collaboration diagrams) and statecharts for concurrency, interaction and behavioural aspects; and deployment diagram for deployment. A textual specification language, *Object Constraint Language* (OCL), is also part of UML for writing constraints on the diagrams and pre- and post-conditions of operations and methods.

The multi-view and multi-notational aspect of UML has an obvious good purpose to allow the split of an overall system model into several views and decompose it into chunks of manageable size. Each single view focuses on a different aspect and this will ease for analysis and understanding. This decomposability of the model enable the development team to split the work of producing models among different people. It is also important for tool support as it would be more difficult for a tool to deal with one large and complex model. UML also intends to support modelling a system at different levels of abstraction. However, without clear means for information hiding, this promise is not as obvious as the one discussed earlier and we need more effort to make UML support model transformation and refinement better.