An Architecture with Multiple Meta-Levels for the Development of Correct Programs

Barbara Dunin-Kęplicz

Institute of Informatics
University of Warsaw
02-097 Warszawa, Banacha 2
Poland
e-mail: keplicz@mimuw.edu.pl

Abstract

In this paper we design a multi-meta-level compositional architecture for correct programs development. In this architecture an object level, describing an application domain, together with a meta-level, representing the semantics of a programming language, and a meta-meta-level, reflecting the adopted methodology, provide a specification of a generic system supporting the user in the process of correct programs construction. The ideas reported in this paper are illustrated in a prototype version of the system, designed for Dijkstra's guarded command programming language.

Keywords: multi-level compositional architecture, formal program development, specification and verification, programming in logic.

1 Introduction

Most software systems designed to help the user in a program derivation are so large that they need the support of a rigorous methodology. A precise specification of methodological assumptions enables one to describe a correct program development process in formal terms. Our goal in this paper is to design an architecture which gives rise to a system which is parametric w.r.t. methodology, programming language and application domain. Distinguishing and separate treatment of these three aspects of program construction is the key point of the reusability of the system.

This work has been partially supported by grants: KBN 2 1199 9101 (Polish Research Commitee), CRIT-1 (CEC).
The expressive power needed to cover various methodological approaches seems to be hard to realize without research in this field; different methodological frameworks need different tools to formalize them. However it is generally accepted that the most conceptually natural programming style is top-down program development. For this reason our general framework will be characterized through the two main principles, namely:

- the *step-wise refinement* technique.
- *verifying* the correctness of the program *during* its development.

The above methodological perspective can be expressed via an adequate parametrization of various aspects of the program construction.

The first, very intuitive, principle is the realization of an analytic design scheme characterized in [9] (p.139):

The analytic design consists of a systematic application of the following reasoning:

1. Given a problem \( P \), is it possible to express its solution in a reasonably concise fashion using primitive notions of the linguistic level at which we want to program? If yes – write the program, if not – invent notions \( P_1, \ldots, P_n \), such that
   (a) each of the notions \( P_1, \ldots, P_n \) is well specified,
   (b) using the notions according to their specification it is possible to write a satisfactory program for problem \( P \).

2. Consider each of the notions \( P_1, \ldots, P_n \) in turn as a new problem and repeat the reasoning.

This process continues until all invented, intermediary notions are implemented in terms of primitive notions.

Applying this scheme we can show that a program may be constructed by creating a sequence of refinements. Each "notion" may be treated as a well-specified module: let the *symbolic module* \( \Pi_i \) correspond to the notion of \( P_i \) (for \( i = 1, \ldots, n \)). In subsequent steps, solving the problem amounts to consecutive development of particular symbolic modules. During the refinement of the program, on any level its correctness may be formally verified. In other words, the notion of symbolic module enables one the on-line verification of the program correctness.