Two Approaches for Developing Generic Components in VHDL

Vytautas Stuikys, Giedrius Ziberkas, Robertas Damasevicius, Giedrius Majauskas
Kaunas University of Technology, Software Engineering Department
Studentu 50, 3031-Kaunas, Lithuania
{stuik,ziber,damarobe,giedmaja}@soften.ktu.lt

Abstract

We consider the one- and two-language approaches (1LA & 2LA) for developing generic components (GCs) for VHDL generators. By 1LA & 2LA we mean a generalization using "pure" VHDL, or using the VHDL abstractions mixed with Open PROMOL, the external scripting language we have developed for building GCs and generators, respectively. We present the evaluation of both approaches.

The 1LA or 2LA is a methodology consisting of these phases: 1) formulation of requirements; 2) domain analysis (DA); 3) analysis and validation of requirements; 4) mapping of requirements into the generalized specification; 5) testing and certification; 6) documentation of GCs.

Development of a GC (see Figure) either by the 1LA or 2LA is a procedure of mapping a given set of functional requirements into two abstract descriptions: the generalized interface and generalized functionality. We usually generalize a component using its *existing instance(s)* gained from DA. We measure the extent of the generalization by wideness of requirements. Approaches differ in the phases 4 & 5 only.

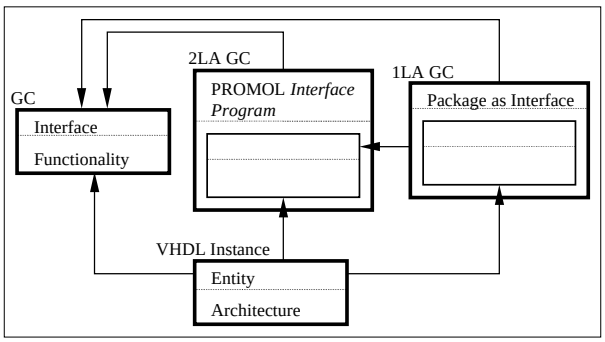

Figure. The GC model in our approaches

The 1LA requires one development environment only ("pure" VHDL). A GC developed by the approach has a simpler validation, no naming problem. However, the approach suffers from the limitations of synthesis tools, the overgeneralization and modifiability problems.

The 2LA is based on the experimental scripting language Open PROMOL, which implements the *open set of external functions*. The approach consists of: 1) mapping of the functional requirements into a set of external parameters; 2) specifying the feasible parameter values; 3) coding of the GC interface; 4) developing of the GC specification body using a functionality coded in VHDL and *promol*-functions.

PROMOL has several advantages in comparison with other scripting languages. The miscellaneous specific functions give the power when implementing the proposed GC model. The feature of using the target language (TL) code mixed with *promol*-functions as an argument of a *promol*-function allows a deep implicit nesting of the functions and, therefore, provides the capabilities of the flexible TL code modifications.

From the user's viewpoint, the 2LA has no overgeneralization problem (user always works with instances). The 2LA is more adaptable to the synthesis limitations and has a higher flexibility for modifications. Additionally, the 2LA does not depend upon the TL. Weaknesses of the 2LA are two development environments (PROMOL & VHDL), the naming, pretty-printing problems, and more complicated validation.

The 1LA & 2LA support the hierarchical generalization, i.e., composing GCs from the smaller ones. We illustrate the capabilities of the approaches by developing GCs, which instances may be met either in the typical or specific designs, e.g., using discrete transformations.

We summarise the main benefits of the approaches as follow:

- The 1LA is superior to the 2LA with respect to black-box reuse. The 2LA is superior to the 1LA with respect to white-box reuse.
- Both approaches ensure a significant reduction in number of the library components, and lead to a higher flexibility when combined together in the VHDL generator model.
- We suggest the virtual component-based generator model. It includes the GC library, PROMOL processor, specification module, and VHDL compiler.