LePUS3: An Object-Oriented Design Description Language

Epameinondas Gasparis\textsuperscript{1}, Jonathan Nicholson\textsuperscript{1}, and Amnon H. Eden\textsuperscript{1,2}

\textsuperscript{1}The Two-Tier Programming Project, Department of Computing & Electronic Systems, University of Essex, United Kingdom
\textsuperscript{2}Centre for Inquiry, Amherst, NY, USA

1 Introduction

LePUS3 [1] (lepus.org.uk) is a logic, visual, object-oriented Design Description Language: a formal specification language designed to capture and convey the building-blocks of object-oriented design. LePUS3 minimal vocabulary constitutes of abstraction mechanisms that can specify effectively and precisely design patterns and the design of Java\textsuperscript{TM} (C++, Smalltalk, etc.) programs at any level of abstraction.

LePUS3 was tailored to integrate the strength of other specification and modelling notations, most notably UML, but it is unique in addressing the combination of the following concerns:

- **Rigour.** LePUS3 is a logic visual language: a chart stands for a formula in an axiomatized theory in the classical first-order predicate calculus.
- **Parsimony & scalability.** LePUS3 offers powerful abstractions: charts scale well and do not clutter with the size of the program.
- **Minimality.** LePUS3 vocabulary (Figure 1) is minimal, consisting of 15 tokens.
- **Decidability & verifiability** [2]. Consistency between a given specification (a chart) and an implementation (a Java program) can be verified by a button-click.
- **Program Visualization.** Charts modelling Java programs can be reverse-engineered from source code.

To emphasize practicalities, we focus on tool support [3][4] in specifying, (automatically) verifying, and visualizing Java programs in LePUS3.

![Fig. 1. LePUS3 vocabulary](image-url)

2 Visualizing Programs

We take program visualization to be a tool-assisted process of discovering some of the building-blocks in the design of programs, and charting them at the appropriate level of abstraction. The motivation is usually understanding and re-engineering large, complex, and inadequately documented programs. We call the approach we take “Design Navigation” [5]: a user-guided process of reverse-engineering LePUS3 charts from the source code of arbitrarily-large Java™ programs. Design Navigation in package java.io (Java™ Software Development Kit 1.6) is demonstrated below using the Two-Tier Programming Toolkit [3][4].

After analyzing package java.io, Design Navigation commences from the Top Chart (Chart 1), the most abstract representation of any Java program. Chart 1 depicts the set of all static types (classes, interfaces, etc.) in java.io as a ‘1-dimensional class’. From Chart 1, Design Navigation proceeds by a user-guided, tool-assisted step-wise application of concretization (‘zoom-in’) and abstraction (‘zoom-out’) operators (left panel, Chart 1). At each step, the Two-Tier Programming Toolkit discovers inheritance class hierarchies (triangles), sets of classes (shaded rectangles), sets of dynamically-bound methods (superimposed ellipses), and correlations amongst them (arrows), visualized using LePUS3 terms and predicates (Figure 1).

For example, Chart 2 offers a birds-eye view of the Closeable class inheritance hierarchy in java.io, generated in a short sequence of concretizations of Chart 1.