Presented in the 5th International Conference on the Theory and Applications of Diagrams—DIAGRAMS, Herrsching, Germany, 19–21 Sep. 2008. Appeared in Gem Stapleton et al. (eds.) Lecture Notes in Artificial Intelligence Vol. 5223, pp. 364–367. Berlin: Springer, 2008.

The work described here has subsequently been significantly revised and expanded in:

Amnon H. Eden, with contributions from Jonathan Nicholson. Codecharts: Roadmaps and Blueprints for Object-Oriented Programs. Hoboken, New Jersey: Wiley-Blackwell, 2011. www.lepus.org.uk/ref/book/

LePUS3: An Object-Oriented Design Description Language

Epameinondas Gasparis ([1]) Jonathan Nicholson ([1]) Amnon H. Eden ([1,2])

([1]) The Two-Tier Programming Project, Department of Computing & Electronic Systems, University of Essex, United Kingdom. ([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™ (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.

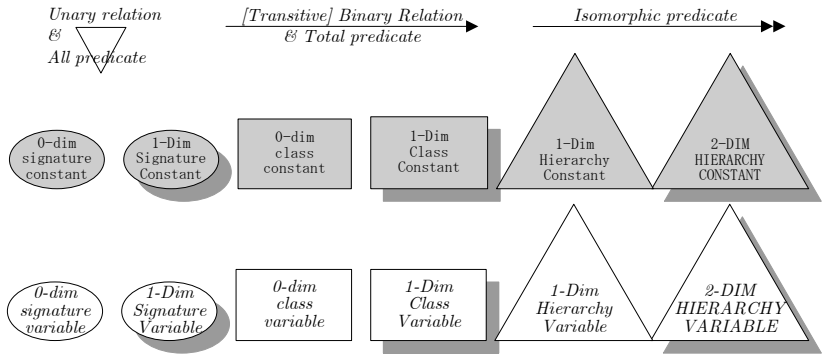

Figure 1. LePUS3 vocabulary

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].

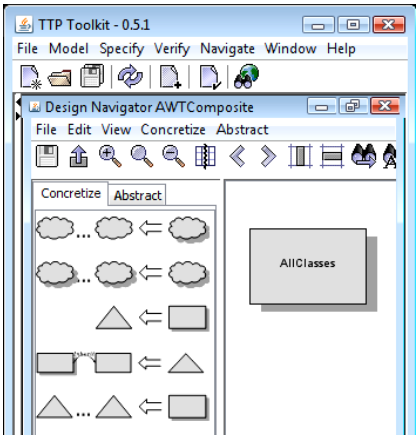

Chart 1. All the classes in `java.io`

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.

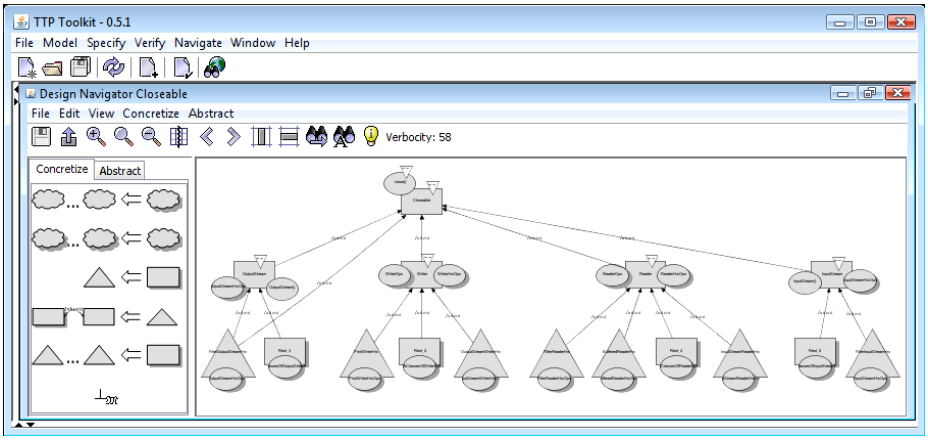

Chart 2. The `Closeable` hierarchy in `java.io`

3. Specifying Programs and Patterns

We take the goal of visual specification to be that of articulating non-functional requirements on object-oriented design in a precise visual language *at the appropriate level of abstraction*. LePUS3 can be used to specify generic design motifs, such as the Composite design pattern (Chart 3), as well as the overall design of a specific implementation, such as package `java.awt` (Chart 4).

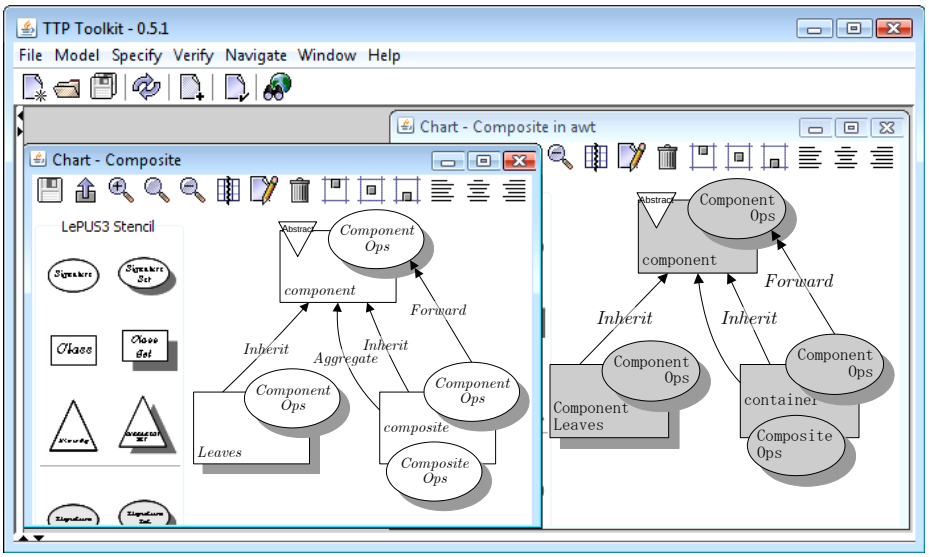

Chart 3. The Composite pattern **Chart 4.** `java.awt` (selected elements from)

The similarity between Chart 3 and Chart 4 is not accidental: package `java.awt` was designed to implement the Composite design pattern [6]. In LePUS3, this intent is made explicit by a logic *assignment* (Figure 2): a mapping which indicates where (and how often) the design pattern is supposed to be implemented. The Two-Tier Programming Toolkit can be used not only to specify assignments but, as we show in the next section, also verify them at a click of a button.

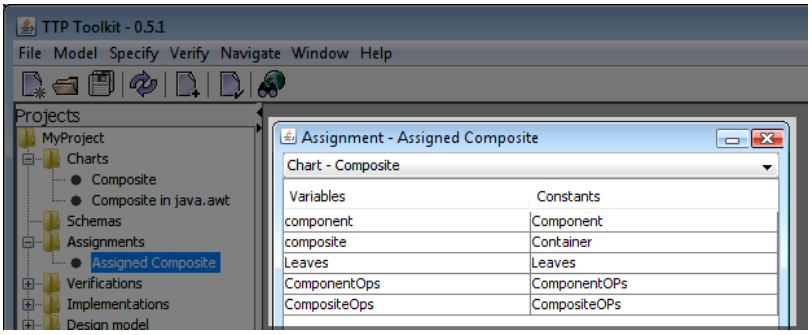

Figure 2. Assignment from the Composite design pattern (Chart 3) to `java.awt` (Chart 4)

4. Automatically Verifying LePUS3 Specifications

By *verification* [2][7] we refer to the rigorous, conclusive, and decidable process of establishing or refuting consistency between a specification (chart) and a program. Fully automated verification is possible in principle since LePUS3 is decidable. It is also possible in practice, as proven by the Two-Tier Programming Toolkit, which implements the verification algorithm. For example, the toolkit can verify by a button-click that package `java.awt` *satisfies* the Composite design pattern as specified in Chart 3, delivering its results in seconds (Figure 3). If verification fails, the toolkit indicates clearly which parts of the specification were violated to allow programmers to fix such inconsistencies.

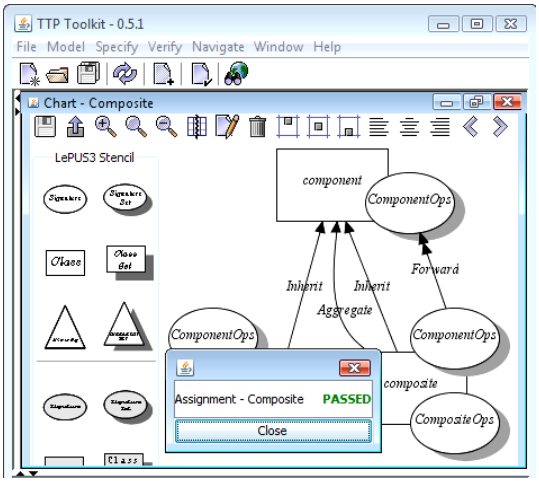

Figure 3. The TTP Toolkit verification results

Acknowledgments: This work was partially funded by EPSRC. The authors wish to thank Ray Turner and Rick Kazman for their numerous contributions to this project.

References

[1] A.H. Eden, E. Gasparis, J. Nicholson. "LePUS3 and Class-Z Reference Manual." University of Essex, Tech. Rep. CSM-474, ISSN 1744-8050 (2007).

[2] J. Nicholson, A.H. Eden, E. Gasparis. "Verification of LePUS3/Class-Z Specifications: Sample models and Abstract Semantics for Java 1.4." University of Essex, Tech. Rep. CSM-471, ISSN 1744-8050 (2007).

[3] E. Gasparis, A.H. Eden, J. Nicholson, R. Kazman. "The Design Navigator: Charting Java Programs." *30th Int'l Conf. Software Engineering* (10–18 May 2008), Leipzig, Germany.

[4] http://ttp.essex.ac.uk/.

[5] E. Gasparis, A.H. Eden. "Design mining in LePUS3/Class-Z: search space and abstraction/concretization operators." University of Essex, Tech. Rep. CSM-473, ISSN 1744-8050 (2007).

[6] E. Gamma et al. *Design patterns: elements of reusable object-oriented software*. Boston: Addison-Wesley, 1995.

[7] J.M. Wing. "A Specifier's Introduction to Formal Methods." *Computer* 23:8 (1990), pp. 8–24.