The Design Navigator: Charting Java Programs

Epameinondas Gasparis
Dept of Computing and Electronic Systems, University of Essex, UK
http://ttp.essex.ac.uk

Amnon H. Eden
Dept of Computing and Electronic Systems, University of Essex, UK and Center for Inquiry, Amherst, NY, USA

Jonathan Nicholson
Dept of Computing and Electronic Systems, University of Essex, UK

Rick Kazman
Software Engineering Institute, Carnegie-Mellon University and University of Hawaii, USA

ABSTRACT
The Design Navigator is a semi-automated design mining tool which reverse engineers LePUS3 design charts from Java™ 1.4 programs at any level of abstraction in reasonable time. We demonstrate the Design Navigator's step-wise charting process of Java Foundation Classes, generating decreasingly abstract charts of `java.awt` and discovering building-blocks in its design.

Categories and Subject Descriptors
D.2.7 [**Software Engineering**]: Distribution, Maintenance, and Enhancement - *restructuring, reverse engineering and reengineering.*

General Terms: Design, documentation, languages

Keywords: Reverse engineering, design mining, software visualization, software modelling, object-oriented design

1. INTRODUCTION
It is a scenario that every experienced programmer dreads: use a large program whose documentation is inaccurate, incomplete or altogether nonexistent [5]. Commercial reverse engineering tools can generate unreadable diagrams cluttered with hundreds or thousands of visual tokens. In contrast, the Design Navigator analyzes any Java™ 1.4 program, discovers its conceptual building-blocks, and charts (any part of) it at any level of abstraction in the entire spectrum between maximum information (a minimally-abstract or 1:1 map) and minimal information (a maximally-abstract or 1:∞ map).

The literature demonstrates that design mining is difficult but not impossible [5][6][7]. The Design Navigator's approach to the problem is distinguished from most existing tools by the following:

- *User-guided*: Charts are generated interactively using zoom-in ('concretization') and zoom-out ('abstraction') operations
- *Formal*: Charts are statements in LePUS3 [1], a mathematically defined object-oriented architecture description language
- *Programming-language independent*: The Design Navigator is not tailored to a specific language (only its analyzer is)

We demonstrate the capabilities of the Design Navigator in charting a small subset of the Java Foundation Classes 1.4 (package `java.awt`), henceforth *JFC*. We also discuss how the application of abstraction or concretization operators promotes program understanding by gradually discovering the building-blocks in the design of Java™ programs that are otherwise hard to spot.

ICSE'08, May 10–18, 2008, Leipzig, Germany.
ACM 978-1-60558-079-1/08/05.

2. DESIGN NAVIGATION
Design Navigation is a user-guided approach to design mining which generates, on demand, visual representations (LePUS3 charts) of programs. Since LePUS3 is mathematically defined [1], the hypothetical 'set of charts of *JFC*' (formally: the set of charts that *JFC implements*) is also well-defined (Figure 1):

$$Charts(JFC) \triangleq \{C \mid JavaSemantics(JFC) \models C\}$$

Theoretical analysis of this set [4] reveals that it has the mathematical properties of a *lattice* (Figure 1): a diamond-shaped grid of interconnected nodes (charts). At the bottom is the chart with maximum information about *JFC*, and at the top is the chart with minimal information about *JFC*.

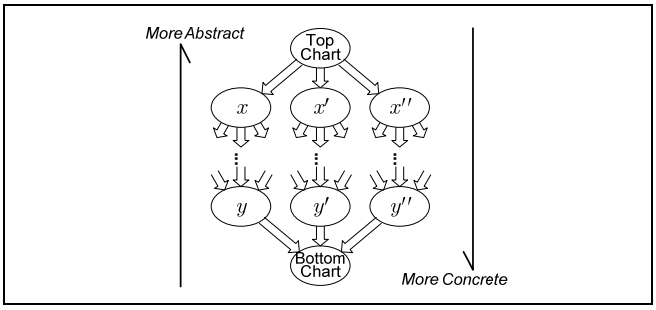

Figure 1 – $Charts(JFC)$: The (hypothetical) set of charts of *JFC* is shaped as a mathematical *lattice*

Given this well-defined conceptual framework, Design Navigation in *JFC* is defined as a tool-assisted, process of traversing $Charts(JFC)$ by step-wise application of abstraction/concretization operators. Each Design Navigation step is formally defined as the product of either an *abstraction* ('zooming-out', or 'up' in Figure 1) or a *concretization* ('zooming-in', or 'down' in Figure 1) operator. Since each chart C can be concretized or abstracted in many ways, depending on which [set of] term[s] is concretized/abstracted, each node C in Figure 1 represents a chart in $Charts(JFC)$ which is connected with several nodes above and below it such that—

- a node connected above C represents an *abstraction* of C
- a node connected below C represents a *concretization* of C

3. DESIGN NAVIGATOR
The Design Navigator operates in two stages: at the preliminary stage, it analyzes a Java program and creates a repository of facts about it. At the second stage, demonstrated in the next section, De-

sign Navigation commences, during which the user repeatedly instructs the Design Navigator to apply an abstraction or a concretization operator, producing at each step a new chart, until the desired level of abstraction is reached.

The Design Navigator has been used to reverse engineer various popular class libraries and frameworks such as JGraph, JDOM, JUnit and other packages from the Java™ class library including the Servlet API, packages `java.io` and `java.util`. Even for large programs (a few hundreds of classes), it produces appropriately abstract charts in tens of seconds.

4. CASE STUDY: CHARTING *JFC*

What can we learn about *JFC* using the Design Navigator? We present the LePUS3 charts produced using the concretization operators. These charts have been modified to improve readability and remove some terms (using the Elimination abstraction operator).

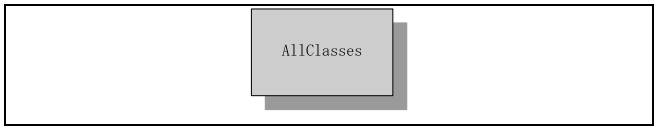

Chart 1 – `AllClasses` stands for the set of all classes in *JFC*

Chart 1, the Top Chart of *JFC*, is generated by the Design Navigator by analyzing *JFC* and only reveals that there exists a set of classes in our program. To discover more about `AllClasses` we apply the Partition concretization operator, producing Chart 2.

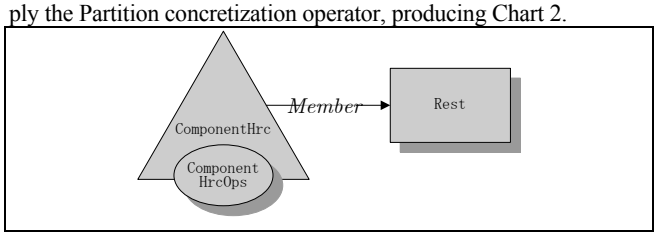

Chart 2 – `ComponentHrc` represents an inheritance hierarchy

In Chart 2 the Design Navigator discovered an inheritance hierarchy in *JFC*, denoted `ComponentHrc`, such that—

- each class in `ComponentHrc` has (at least) one field ('Member') of some class in the set of classes `Rest`
- each class in `ComponentHrc` has a (possibly inherited) set of the methods, each of which has a method signature in `ComponentHrcOps`. In other words, there exists a set of sets of dynamically-bound methods in `ComponentHrc`

To reveal more about the classes and methods in the `ComponentHrc` hierarchy, we apply the Enumeration and Partition concretization operators, resulting in Chart 3, discovering that—

- class `Container` 'aggregates' (has at least one array field holding instances of [subtypes of]) `Component`
- classes `Component`, `Button` and `Scrollbar` define a set of dynamically-bound methods, sharing the same set of method signatures (`ComponentOps2`)
- each method in class `Container` whose signature is in `ComponentOps2` forwards the call to the respective method in class `Component`

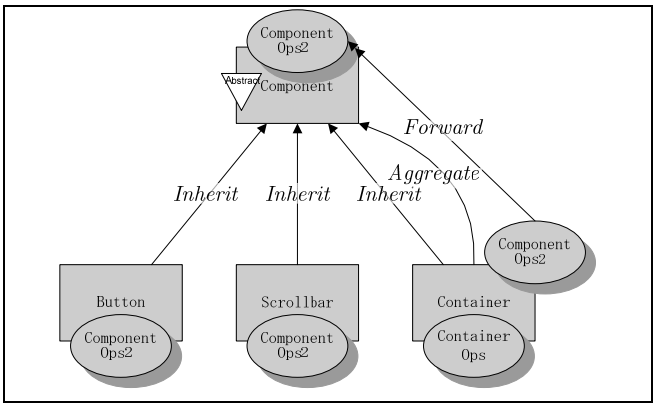

Chart 3 – Charting an instance of the Composite in *JFC*

In fact, Chart 3 completely matches the description of the Composite [3] pattern in LePUS3 [2], convincing us that classes `Button`, `Component`, `Container` and `Scrollbar` participate in an instance of the Composite pattern.

5. CONCLUSION

We have presented the Design Navigator, a tool that allows programmers to reverse engineer concise, precisely defined, appropriately abstract LePUS3 charts from Java™ 1.4 programs in polynomial time, solely by the stepwise application of design navigation operators. The tool gradually discovers some of the building-blocks in the design of a program, bringing to light relationships which otherwise are difficult to detect and charts them at the appropriate level of abstraction. The principles of Design Navigation as demonstrated by the Design Navigator can be useful to many programmers who wish to understand unfamiliar Java™ programs.

6. REFERENCES

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

[2] Eden, A.H., Nicholson, J., and Gasparis, E. 2007. The 'Gang of Four' Companion: Formal specification of design patterns in LePUS3 and Class-Z. Tech. Rep. CSM-472, ISSN 1744-8050, University of Essex. www.lepus.org.uk

[3] Gamma, E., et al. 1995. *Design Patterns: Elements of Reusable Object-Oriented Software*. Reading, Addison-Wesley.

[4] Gasparis E., and Eden, A.H. 2007. Design Mining in LePUS3/Class-Z: Search Space and Abstraction/Concretization Operators. Tech. Rep. CSM-473, ISSN 1744-8050, University of Essex. www.lepus.org.uk

[5] Kazman, R., and Carrière, S. J. 1999. Playing Detective: Reconstructing Software Architecture from Available Evidence. *Automated Software Eng.* 6(2) 107–138.

[6] Murphy, G. C., et al. 2001. Software Reflexion Models: Bridging the Gap between Design and Implementation. *IEEE Trans. Soft. Eng.* 27(4) 364–380.

[7] Storey, M., et al. 2002. SHriMP views: an interactive environment for information visualization and navigation. In *CHI '02 Extended Abstracts on Human Factors in Computing Systems* (Minneapolis, Minnesota). New York: ACM, 520–521.