Green: A Customizable UML Class Diagram Plug-In for Eclipse

Carl Alphonce
Dept. Comp. Sci. & Eng.
University at Buffalo, SUNY
Buffalo, NY 14260-2000
alphonce@cse.buffalo.edu

Blake Martin
Dept. Comp. Sci. & Eng.
University at Buffalo, SUNY
Buffalo, NY 14260-2000
bcmartin@cse.buffalo.edu

ABSTRACT

This poster presents a customizable round-tripping UML class diagram plug-in for Eclipse called Green. While this tool was developed primarily with students and instructors in introductory OO computer science courses in mind, its extensible architecture makes it potentially useful to others.

Green is a flexible tool: each binary class relationship is implemented as a separate plug-in to the basic tool. The set of relationships supported is therefore easily tailored to the needs of the user. More importantly, the semantics of the relationships can be defined to suit the user. The tool can therefore adapt to the needs of its users.

Categories and Subject Descriptors

K.3.2 [**Computers and Education**]: Computer and Information Science Education—*Computer Science Education*; D.2.3 [**Software Engineering**]: Coding Tools and Techniques—*Object-oriented programming*

General Terms

Design

Keywords

Design, Object-orientation, UML, CS1, CS2, Java, code generation, reverse-engineering

1. INTRODUCTION

This poster presents an overview of a simple-to-use round-tripping[1] and customizable UML class diagram plug-in for Eclipse called Green. Green was developed primarily for use in introductory OO computer science courses, but has been designed to grow with students as their skills increase: both the set of class relationships, as well as the semantics of those relationships, is customizable. Green is a plug-in to Eclipse, and provides an extension point through which class relationships are defined, as plug-ins to Green.

[1] *Round-tripping* means that it can both generate code from a class diagram and reverse-engineer a class diagram from code, thereby enabling a complete round-trip from either representation to the other.

2. MOTIVATION

The introductory (object-oriented) computer science sequence which the first author teaches emphasizes the importance of good OO design.[1] Pedagogically we have found that in order to achieve this focus on design it is critical to raise the level of abstraction in the thinking and discourse of our students [5] and make program design a significant part of the assessment of student work.[2]

The course sequence introduces students to good design via immersion in design patterns. Faculty teaching the course have found it invaluable to use UML class diagrams as a language for discussing design with students because these diagrams are more abstract than code, allowing us to ignore irrelevant details and focus on the essence of a design.

We introduce UML class diagrams incrementally. Initially students are told about classes (not interfaces) and one relationship (composition). We employ a very narrow and concrete definition of composition so that students can easily map from code to UML and vice versa. Over time students learn about interfaces and more relationships (association, dependency, generalization and realization). As students become more familiar with these relationships, translating diagrams into Java manually becomes *tedious*, *time-consuming* and therefore *error prone*.

Students are also expected to read and understand code written by others. We feel it is easier to understand the structure of an OO system by way of a class diagram than by inspecting the system's code. Constructing class diagrams by hand also quickly becomes *tedious*, *time-consuming* and therefore *error prone*.

UML should not be simply another notation students need to learn, with no immediate practical benefit: students expect immediate gratification from what they are taught. Because Green supports round-tripping it provides such gratification: once students understand relationship semantics they are free to use Green to manipulate their designs.

The tool is also used to assist in the grading of student work. Students are very perceptive: they notice very well what aspects of their work contribute to their grade. In our experience no matter how much we stress design, if students' submissions are not graded for design students will only pay lip service to it. Grading for design without tool assistance is *tedious*, *time-consuming* and therefore *error prone*.

Use of Green makes grading of student work for design feasible. Teaching assistants are able to recover the design of large projects quickly and easily, and can more readily understand the structure of submissions.

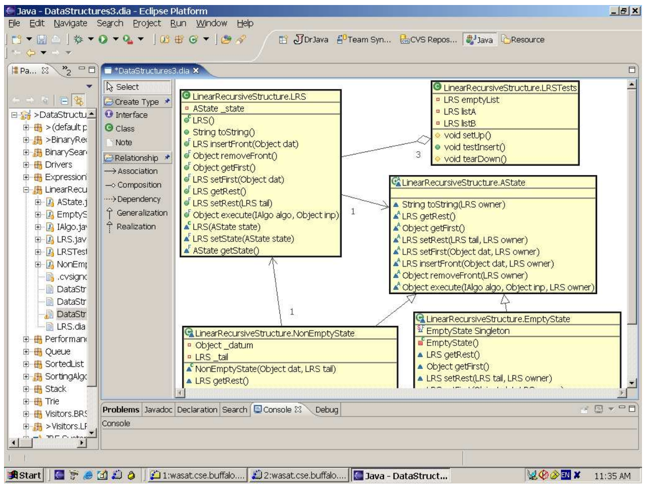
Java - DataStructures3.dia - Eclipse Platform
*DataStructures3.dia
Select
Create Type
Interface
Class
Note
Relationship
Association
Composition
Dependency
Generalization
Realization
LinearRecursiveStructure.LRS
LinearRecursiveStructure.LRSTests
LinearRecursiveStructure.AState
LinearRecursiveStructure.NonEmptyState
LinearRecursiveStructure.EmptyState
Problems
Javadoc
Declaration
Search
Console
Debug

Figure 1: A screen shot of Green

3. TOOL FUNCTIONALITY

As noted, Green is a round-tripping tool: it allows both generation of code from a diagram and reverse-engineering of a diagram from code. Figure 1 shows a screen shot.

Guéhéneuc [3] notes that many class-diagram tools do not synchronize their diagrams with the code those diagrams purport to represent: “Class diagrams . . . are often obsolete — unsynchronised with the *concrete* implementation of programs. . . ” While Green does allow the saving of snapshots of a diagram (for later inclusion in a project description, for example) or to maintain diagram layout information across sessions, the diagram which Green maintains is essentially a view on a collection of Abstract Syntax Trees (ASTs) which the Java Development Tools (JDT) maintains. The diagram is thus sensitive to changes in the AST of any given type, regardless of the source of the modification (the diagram editor or the code editor). Likewise, any modification performed via the diagram changes the AST for the affected type, and is immediately reflected in JDT’s code editor.

The tool allows two modes of reverse generation. In the first the user selects a set of classes and the tool displays a class diagram consisting of those classes, and all the binary class relationships which the tool recovers amongst them. The second is called incremental exploration. In this mode the user explores from a given class (or interface), building a class diagram incrementally by including in the diagram at each step all the immediate neighbors of a selected class (by following those relationships which originate in that class).

4. TOOL ARCHITECTURE

Green’s basic framework is a plug-in to Eclipse. It provides an editor for manipulating class diagrams. Green maintains a UML model, distinct from JDT’s ASTs, which has explicit representations of types (classes and interfaces), relationships (e.g. composition, realization), notes, and location information for each of the preceding elements.

Green provides an extension point by which relationships can be added to the basic framework. The set of relationships available is determined simply by the set of relationship plug-ins which have been installed. A relationship is defined by three main parts. A **recognizer** traverses the AST of a type and identifies relationships of a specific sort (e.g. composition) which originate in that type. The recognizers of the installed relationships are run whenever Green is notified of a change in the AST (Java model). A **generator** traverses the AST of a type and modifies it so that the specific relationship it represents is injected into the AST. A generator is run when the user draws a relationship between two elements (classes or interfaces) in the diagram. A **remover** traverses the AST of a type and modifies it so as to remove that relationship.

Guéhéneuc *et al.* [4] note that different UML tools use different definitions for the same binary class relationships. We note here that rather than impose a single definition of a relationship, Green offers the user the chance to tailor the definition of a relationship so that it meets their needs.

5. CONCLUSION AND FUTURE WORK

Green is a round-tripping UML class diagram plug-in for Eclipse which is customizable to allow users to supply a set of binary class relationships, along with their semantics, which make sense for them. We have found the tool useful in teaching OO programming and design.

The definition of a relationship in three parts is difficult because the semantics of all three components must be the same. Ideally the semantics of a relationship would be specified once, independently of the operation being performed (recognition, generation or removal). We are exploring providing a meta-language for defining relationship semantics in a declarative (rather than a procedural) manner.

In the longer term we plan a design pattern extension. Many design patterns are difficult to distinguish because their static structure is very similar if not identical (e.g. Strategy vs. State). This is a challenging problem, with a rich literature. Adding design pattern support to Green would enhance its usefulness to students.

6. REFERENCES

[1] C. Alphonce and P. Ventura. Object-orientation in CS1-CS2 by design. *ACM SIGCSE Bulletin*, 34(3):70–74, 2002.

[2] C. Alphonce and P. Ventura. QuickUML: a tool to support iterative design and code development. In *OOPSLA ’03: Companion of the 18th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*, pages 80–81, New York, NY, USA, 2003. ACM Press.

[3] Y.-G. Guéhéneuc. A reverse engineering tool for precise class diagrams. In *CASCON ’04: Proceedings of the 2004 conference of the Centre for Advanced Studies on Collaborative research*, pages 28–41. IBM Press, 2004.

[4] Y.-G. Guéhéneuc and H. Albin-Amiot. Recovering binary class relationships: putting icing on the UML cake. In *OOPSLA ’04: Proceedings of the 19th annual ACM SIGPLAN Conference on Object-oriented programming, systems, languages, and applications*, pages 301–314, New York, NY, USA, 2004. ACM Press.

[5] D. Nguyen and S. Wong. OOP in introductory CS: Better students through abstraction. The Fifth Workshop on Pedagogies and Tools for Assimilating Object-Oriented Concepts, *OOPSLA ’01*, 2001.