Two Applications for Transient Views in Software Development Environments

Christoph Daniel Schulze, Miro Spönemann, Christian Schneider, and Reinhard von Hanxleden
Dept. of Computer Science, Christian-Albrechts-Universität zu Kiel, Germany
Email: {cds,msp,chsch,rvh}@informatik.uni-kiel.de

***Abstract*—Pragmatics-aware modeling refers to model-driven engineering with designer productivity in mind. We apply this concept to traditional software development by introducing two exemplary applications for transient views geared at increasing developer productivity: UML class diagram generation and debug state visualization.**

I. Introduction

Pragmatics-aware modeling [1] aims to increase designer productivity by taking a human-centric approach of allowing the modeler to focus on the model, and by having the modeling tool create customized views automatically as required. This is enabled by the *transient views approach*, which allows the efficient, automatic creation of graphical views, customized to specific requirements [2].

However, transient views cannot be applied only to model-based development. Developers using traditional programming languages such as Java can make use of dynamic visualization as well. In this paper, we apply the concept of *pragmatics-aware modeling* to traditional development environments, thereby introducing *pragmatics-aware development*.

Contribution: We present two applications for transient views in traditional development environments: dynamic generation of UML class diagrams allows developers to quickly generate graphical views of an application's structure; debug state visualization enables developers to graphically inspect the state of suspended applications. Both applications were implemented into the Eclipse development environment.

Related work: Eclipse-based tools for generating UML class diagrams include *EclipseUML*[1] and *ObjectAid*.[2] Both tools integrate into the development environment and generate dedicated files that represent the diagrams. Both tools can generate diagrams from Java code, EclipseUML can also generate Java code from diagrams. Both tools focus on persisted diagrams. Several tools exist to visualize data structures. The *Lightweight Java Visualizer* library [3] can be used to programmatically generate graphs that represent Java objects, but does not explicitly allow specialized visualizations for certain classes of objects. *Javavis* [4] is an application that uses diagrams to control and inspect the execution of Java programs, but does not allow specialized visualizations either.

II. UML Class Diagram Generation

UML class diagrams are a an established way to help understand a piece of software by visualizing its structure. Traditional UML modeling tools integrated in development environments usually treat the diagrams as first-class citizens to be explicitly edited and persisted. The approach based on transient views instead generates diagrams on the fly based on a set of classes selected in the development environment. The overhead associated with creating a new diagram file, dragging the relevant classes into it, and positioning them properly is replaced by selecting classes in the IDE and invoking a single command from the context menu. Inheritance relationships and associations are found and displayed automatically.

To implement the transient views approach to class diagram generation, we used the KLighD framework, a part of the KIELER project.[3] We added a simple *diagram synthesis* that turns a selection from the Eclipse Project Viewer into a graph structure. The graph structure is then laid out using the OGDF Planarization algorithm[4] to produce a diagram. A sample diagram generated this way is shown in Fig. 1.

The diagram can be customized by setting a number of options, for example whether all or only explicitly selected members of selected classes should be visualized, whether to visualize the package hierarchy, and whether inheritance and association edges should be visualized.

In addition, we provide a textual language through which the exact content of the diagram and all visualization options can be customized and persisted, if required.

III. Debug State Visualization

One of the common tasks in software development is debugging a fault within an application. This often involves running it in a dedicated debug mode that supports setting breakpoints to suspend and inspect the running application. Eclipse provides a number of views on suspended applications, of which one shows the variables and fields in scope at a given breakpoint. This view allows the programmer to inspect and even change the current values of variables and fields and find out whether they match the expectations. The variables and fields are arranged in a tree view. While this is already very helpful, the object graph is usually not a tree: any given object can be referenced by multiple other objects, and back references are common as well. In these cases, objects appear more than once in the tree, causing it to become increasingly unclear.

Debug state visualization presents the variables and fields as a diagram, as shown in Fig. 2. Also based on the KLighD

[1] http://www.ejb3.org
[2] http://www.objectaid.com
[3] http://www.informatik.uni-kiel.de/en/rtsys/kieler/
[4] http://www.ogdf.net/

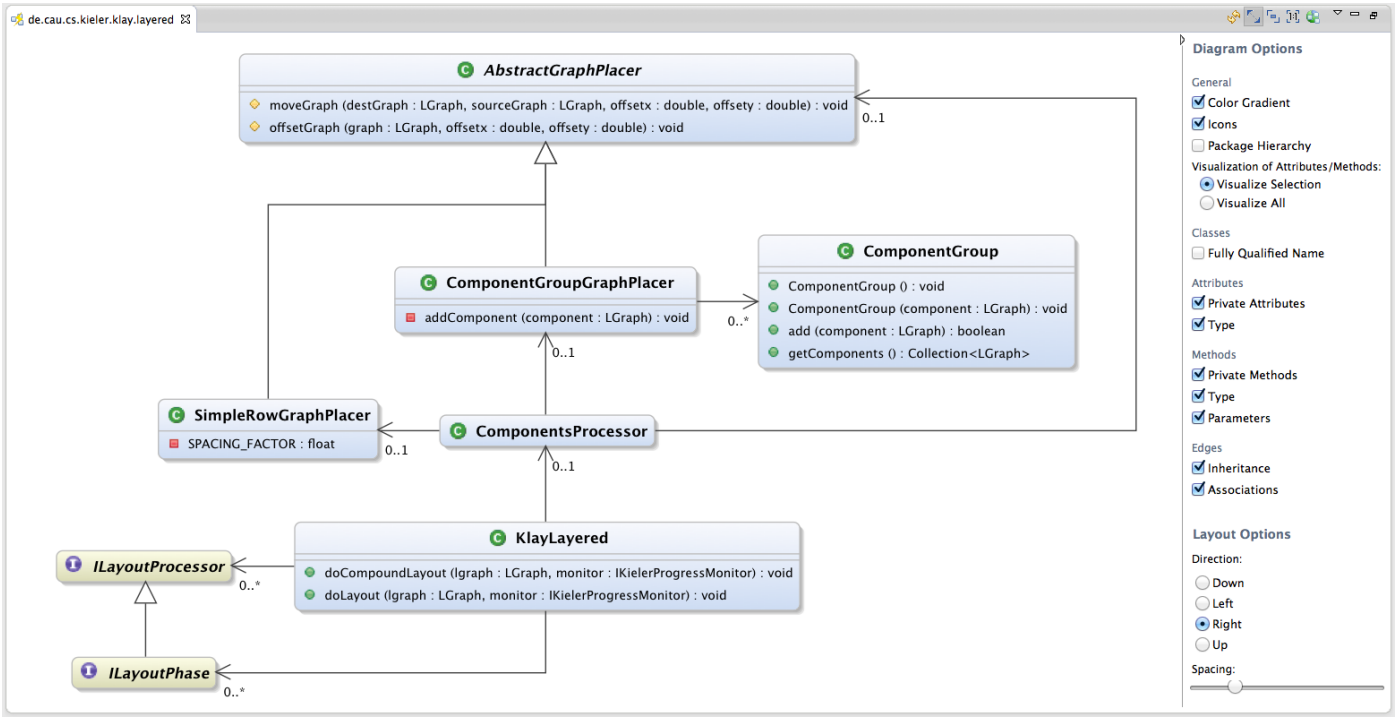

Fig. 1. A class diagram drawn by our implementation of transient views UML class diagram generation.

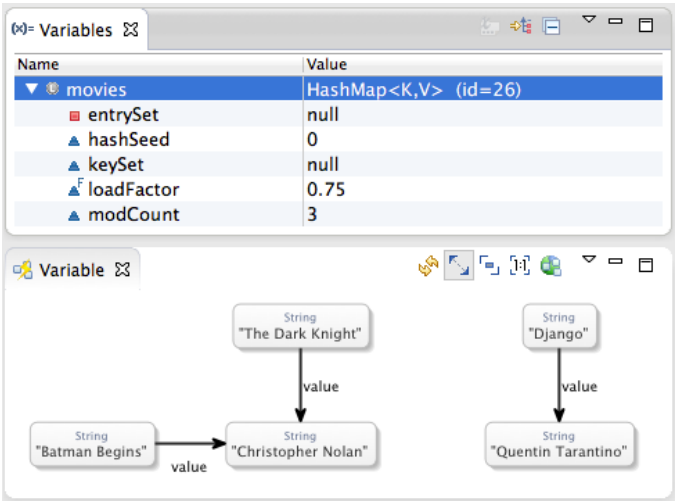

Fig. 2. The debug visualization showing a graphical view of a `HashMap` selected in the variables view provided by Eclipse. Note that two keys map to the same object, which is easy to see in the diagram.

framework, selecting data in the variables view provided by Eclipse opens a view with a node-link diagram representing the objects and their relationships.

The debug state visualization is based on small transformations that specify how to represent instances of a specific class in the diagram. For example, the `StringTransformation` represents a `String` instance as a single node in the diagram labeled with the `String`, while the `LinkedListTransformation` adds a node that represents a `LinkedList`, triggers a transformation for each contained object, and adds links between them. Through an Eclipse extension point, the visualization can easily be extended by adding custom transformations.

IV. Conclusion

With UML class diagram generation and debug state visualization we have shown two applications for transient views in traditional development environments. UML class diagram generation allows to gain a quick overview without the overhead normally associated with UML tools. Debug state visualization offers a visual presentation of the state of suspended applications.

Future work could go into supporting more UML diagrams, such as the automatic generation of sequence diagrams. Regarding debug state visualization, visualizations for many more classes could be added.

References

[1] H. Fuhrmann and R. von Hanxleden, "Taming graphical modeling," in *Proceedings of the ACM/IEEE 13th International Conference on Model Driven Engineering Languages and Systems (MoDELS'10)*, ser. LNCS, vol. 6394. Springer, Oct. 2010, pp. 196–210.

[2] C. Schneider, M. Spönemann, and R. von Hanxleden, "Just model! – Putting automatic synthesis of node-link-diagrams into practice," in *Proceedings of the IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC'13)*, San Jose, CA, USA, 15–19 Sep. 2013, pp. 75–82.

[3] J. Hamer, "A lightweight visualizer for Java," in *Proceedings of the third program visualization workshop, Research Report CS-RR-407*. Department of Computer Science, University of Warwick, 2004, pp. 54–61.

[4] R. Oechsle and T. Schmitt, "JAVAVIS: Automatic program visualization with object and sequence diagrams using the Java debug interface (JDI)," in *Software Visualization*, ser. Lecture Notes on Computer Science. Springer, 2002, vol. 2269, pp. 176–190.