Hierarchy based 3D Visualization of Large Software Structures

Michael Balzer and Oliver Deussen

University of Konstanz, Germany

ABSTRACT

Modern object-oriented programs are hierarchical systems with many thousands of interrelated subsystems. Visualization helps developers to better comprehend these large and complex systems. This work presents a three-dimensional visualization technique that represents the static structure of object-oriented software using distributions of three-dimensional objects on a two-dimensional plane. The visual complexity is reduced by adjusting the transparency of object surfaces to the distance of the viewpoint. An approach called Hierarchical Net is proposed for a clear representation of the relationships between the subsystems.

CR Categories: D.2.7 [Software Engineering]: Distribution, Maintenance, and Enhancement—; I.3.8 [Computer Graphics]: Applications

1 INTRODUCTION

Software systems belong to the most complex artifacts. In many domains of applications, object-oriented systems with millions of lines of code and many thousands of interrelated components are constructed. Typically, they run through a long evolution process, and the expenses for maintenance and reengineering by far surpass the cost of the original design and implementation. Visualization can serve in maintenance and reengineering of software and help to comprehend existing software more efficiently and accurately.

There are many visualization approaches showing different aspects of programs, like the static structure, the runtime behavior, the evolution, or the development process [1, 2]. In this work we focus on the visualization of the static structure, an aspect that is commonly used in the comprehension, quality assessment, and reengineering of large software systems.

Our approach resulted from an exploratory study of how to visualize the static structure of real-world software systems with the landscape metaphor. They combine three-dimensional images of landscape elements, customized layouts, and hierarchical interconnection networks, to represent program entities, their hierarchy, and their relationships. Dynamic transparencies enable the viewer to move seamlessly between abstract and detailed views.

2 A MODEL OF OBJECT-ORIENTED SOFTWARE

The models distinguish four types of software entities: packages, classes, methods, and attributes. Each package can contain other packages and classes, and each class can contain methods and attributes. In Java and other object-oriented programming languages, classes may also contain other classes. Because the contained classes are mostly very simple and tightly coupled to their containing classes, each class is collapsed with their contained classes. The gain of simplicity (in the schema and, more importantly, in the visualizations) through this collapsing by far outweighs the minor loss of precision. Besides containment, the models distinguish three other kinds of relationships between the entities: classes can inherit from other classes, methods can call methods, and methods can access attributes. The schema of the models is shown in Figure 1.

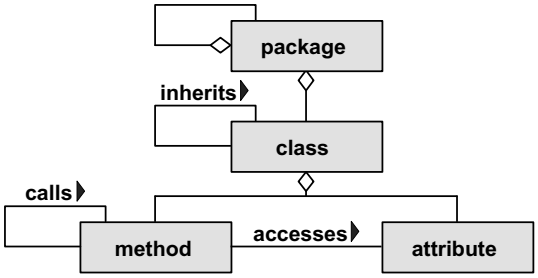

Figure 1: Schema for models of object-oriented software systems

Such models can be automatically extracted from the source code of object-oriented software systems. In our experiments, we used the tools SNiFF+ [7] and Sotograph [6] to extract graph models from Java programs. The extracted graphs are stored in files in Rigi Standard Format [8]. Through the separation of extraction from visualization, and the use of a standard exchange format for graphs, it is possible to visualize data from many sources. In particular, they can be applied to software in any programming language for which an appropriate extractor is available.

3 HIERARCHY BASED LAYOUT OF ENTITIES

The layouts in this work are based on the hierarchy of packages, classes, methods and attributes in the visualized software system. The hierarchy of packages, which can be arbitrarily deep, is represented by nested hemispheres. The outermost hemisphere stands for the root of the package hierarchy. It contains hemispheres which represent the packages that are directly contained in the root package. These hemispheres for the second level packages again contain hemispheres for the third level packages, and so on. The size of the hemispheres is adjusted to the number of methods and attributes contained in the package and all sub-packages. After arranging the packages, the positions of the classes within the center of the according packages are defined. Each class is represented as a circle with a surface area related to the number of contained methods and attributes. Within these circles the methods and attributes are positioned as simple box objects. Figure 2 illustrates the result of this arrangement pattern.

As a tool for the generation of this distributions we use a relaxation based on Voronoi diagrams [3, 4]. Thereby all objects are trying to expand their size as much as possible under the constraint that the relation between their sizes is fixed.

4 HIERARCHICAL NETS

Aside from the hierarchy of the entities, the relations between the entities are an important part of the structure of software systems. If these relations would be represented as simple direct line connections between the entities of a given two-dimensional level, very unclear representations with many overlappings and occlusions would result, which make a differentiation and a closer investigation of the individual relations almost impossible.

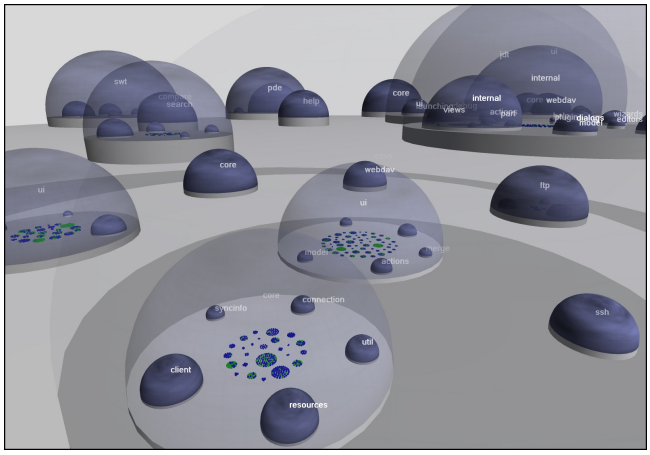

Figure 2: Hierarchy based 3D representation of a software system

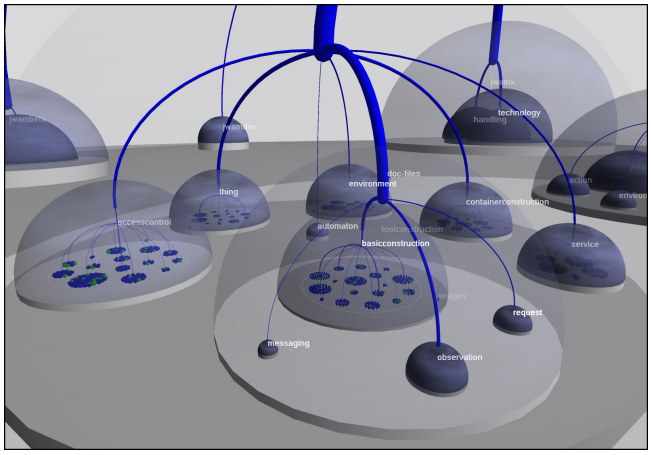

Figure 3: Visualization of entity relations using a Hierarchical Net

We propose a solution called Hierarchical Net. Thereby the relations are routed according to the hierarchy levels of the software entities. For example, if a relation exists between a class *X* in package *A* and a class *Y* in package *B*, and furthermore, the packages *A* and *B* are contained in package *C*, then the connection is routed from class *X* to package *A*, to package *C*, then to package *B*, and lastly to class *Y*. For this purpose a point is defined above every object, where the connections of the objects of the lower hierarchy levels are combined and forwarded to the next level. This point always rests within a fixed relative distance above the center of the considered object. Since the objects at higher hierarchy levels are larger, this results in a three-dimensional tree of connections, as shown in Figure 3.

The type of the relations is shown by the color of the connections. Relations of the same type, and with the same start and end points, are combined to one connection. Thereby their quantity is mapped to the size of the new connection, so that thicker connections stand for a larger quantity of represented relations.

In order to analyze the relations the user can control the visualization. The first possibility is to select only specified types of relations, e.g. the user can solely view all inheritances. The second is to choose an entity, whereby a list with all connected relations is presented, and additionally only relations connected to this entity are drawn in the visualization. These two alternatives enable the better traceability of the relations.

5 Utilizing Transparency for Level of Detail

As already explained, nested hemispheres are used for the arrangement of packages. Pursuant to [5] the representation of the hemispheres takes place with the help of transparencies. Thus a view into the system is possible, while at the same time the presented amount of information is reduced. If the surfaces of the hemispheres would be completely transparent, i.e. only the silhouettes are drawn, it would be difficult to interpret the visualization due to the overabundance of information. If the surfaces were perfectly opaque, objects within or outside the presently viewed packages would be invisible. The use of transparencies solves both problems.

Contrary to [5] the degree of transparency is not fixed in advance, but adapts dynamically to the position of the viewer. If the distance of the viewpoint to a hemisphere is more than five times the hemisphere's radius, then the hemisphere is drawn perfectly opaque. If the distance of the viewpoint is less than two times the hemisphere's radius, then the hemisphere is completely transparent. Between these two distances, seamless cross fading takes place.

Fading out distant levels allows to clearly represent also scenes with a very deep hierarchy. This level of detail technique also improves the rendering performance of the visualization, because the inside of completely opaque hemispheres does not have to be drawn, and normally more than 90 % of all hemispheres are opaque.

6 Conclusion and Future Work

It is significant to note that the presented visualization is an interactive real-time application, which runs on a workstation with 3 GHz and a Nvidia GeForce 5800 graphics adapter with 20 to 60 frames per second.

In future works, more possibilities of the landscape metaphor in the context of software visualization are to be examined. One direction will be the examination of layout methods which are not only based on the hierarchy of the software system, but also involve the relations in the layout generation process. The information density in the visualizations can be further improved by mapping values of software metrics on objects in the visualization. For example, the height of the objects that represent methods could be proportional to the size of the methods, measured by the number of lines of code.

References

[1] *Proceedings of the 1st International Workshop on Visualizing Software for Understanding and Analysis (VISSOFT)*. IEEE Computer Society, 2002.

[2] *Proceedings of the ACM Symposium on Software Visualization (SOFTVIS)*. ACM, 2003.

[3] Oliver Deussen, Stefan Hiller, Cornelius van Overveld, and Thomas Strothotte. Floating points: A method for computing stipple drawings. *Computer Graphics Forum*, 19(3):40–51, 2000.

[4] Kenneth E. Hoff, John Keyser, Ming C. Lin, Dinesh Manocha, and Tim Culver. Fast computation of generalized Voronoi diagrams using graphics hardware. In *Proceedings of the 26th Annual Conference on Computer Graphics (SIGGRAPH)*, pages 277–286. ACM, 1999.

[5] Jun Rekimoto and Mark Green. The Information Cube: Using transparency in 3d information visualization. In *Proceedings of the 3rd Annual Workshop Information Technologies & Systems (WITS)*, pages 125–132, 1993.

[6] Software-Tomography GmbH. http://www.softwaretomography.com.

[7] Wind River Systems Inc. http://www.windriver.com.

[8] Kenny Wong. *Rigi User's Manual, Version 5.4.4*, 1998. http://ftp.rigi.csc.uvic.ca/pub/rigi/doc/.