First publ. as: Paper / Joint Eurographics - IEEE TCVG Symposium on Visualization (VisSym), May 19-21, 2004, Konstanz

Software Landscapes: Visualizing the Structure of Large Software Systems

Michael Balzer,[1] Andreas Noack,[2] Oliver Deussen,[1] Claus Lewerentz[2]

[1] Department of Computer and Information Science, University of Konstanz, Germany
[2] Software Systems Engineering Research Group, Technical University Cottbus, 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 paper presents a three-dimensional visualization technique that represents the static structure of object-oriented programs using landscape-like distributions of three-dimensional objects on a two-dimensional plane. The familiar landscape metaphor facilitates intuitive navigation and comprehension. 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.

Categories and Subject Descriptors (according to ACM CCS):
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 go 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 to comprehend existing software more efficiently and accurately.

Software visualization, as a subfield of information visualization, is the use of various forms of imagery to facilitate the understanding of software systems. There are many visualization approaches showing different aspects of programs, like the static structure, the runtime behavior, the evolution, or the development process [Vis02, Sof03]. 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.

It is central to all visualizations that the visual characteristics of the emerging images are interpretable with regard to the characteristics of the visualized information. If information is spatially meaningfully arranged and graphically represented, abilities of the human perception contribute to the process of understanding. The use of metaphors from the physical world promises to make this process particularly intuitive and effective, because it allows the viewer to transfer existing perceptual abilities to the comprehension of the visualization. We chose the landscape metaphor because it is familiar and suites the hierarchical structure of software.

Software Landscapes 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.

Before the main part, we discuss related work on software structure visualization in Section 2, and detail the underlying model of software systems in Section 3. A description of Software Landscapes is given in Section 5. It is preceded by a discussion of the landscape metaphor in Section 4.

2. Previous Work

Currently, the most popular graphical language in software engineering is the Unified Modeling Language [Obj03]. In UML, the static structure of a system is modeled by class diagrams. Classes can be grouped to packages to obtain diagrams at a higher level of abstraction. UML – like other diagram notations – includes no advanced graphics and visualization techniques. On the one hand, this facilitates the drawing of diagrams by humans and the representation of diagrams in paper documents. On the other hand, it decreases information density and control over the level of abstraction, which limits scalability.

In tools that automatically create visualizations of structural software models, syntactical restrictions to allow easy drawing by humans are irrelevant. Usually, such tools present software entities and their relations as graphs. Rigi [MOTU93], an early and very influential tool for software structure visualization, still creates simple box-and-line diagrams. However, it provides extensive navigation facilities that allow the user to create views of different parts of the visualized system on different levels of abstraction. The more recent tools SHriMP [SM95] and Portable Bookshelf [FHK*97] provide fisheye views of nested graphs. This technique presents at the same time the details and the context of a part of the visualized system.

The use of the third dimension promises an increased information density and larger degree of freedom for graph layouts. After the introduction of three-dimensional graph layouts to software structure visualization by Koike [Koi92] and Reiss [Rei95], many different ways of using the third dimension were explored in systems like Narcissus [HDWB95], NestedVision3D [PFW98], ArchView [FdJ98], and CrocoCosmos [LN03]. An interesting aspect of Narcissus and NestedVision3D, is the use of surfaces with variable transparency to adapt the amount of details shown.

The results of empirical studies that compare the effectiveness of two-dimensional and three-dimensional graph layouts are mixed: In some studies, 3D visualizations outperformed 2D visualization (e.g. [WF96]), other studies yielded the opposite result (e.g. [WC99]). In our experience with 3D layouts of large graphs, individual objects are often occluded and therefore barely recognizable, and orientation is sometimes intricate.

So called 2.5-dimensional visualizations show three-dimensional objects arranged on a two-dimensional surface, similar to landscapes. This approach combines the good orientation and overview of 2D visualizations with a high information density, and exploits the same perceptual abilities that we use in our physical environment. Information retrieval systems based on the landscape metaphor [Cha93, DHJ*98, CP01] mostly visualize data that has different characteristics than software structures, for example non-hierarchical data. THEMA [Plo97], Software World [KM00], and Component City [CKTM02] use an urban metaphor to visualize software systems. However, only few software entities are shown and the graphical representations are very simple. Neither the use of advanced visualization techniques nor the scalability to programs of realistic size are addressed in these works.

3. A Structural Model of Object-Oriented Software

A structural model of an object-oriented software system describes the system's entities, the containment hierarchy of the entities, and the relationships between the entities. Formally, our models are nested directed graphs, where the nodes of the graph correspond to the software entities, the edges correspond to the relations, and the inclusion tree corresponds to the containment hierarchy. The schema of the models is shown in Figure 1.

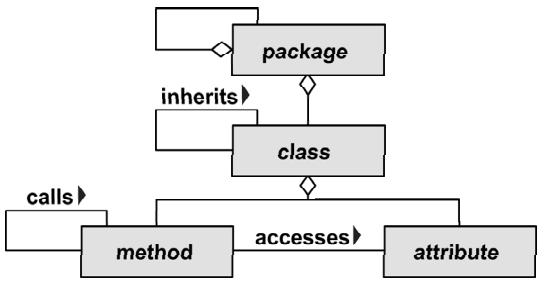

Figure 1: *Schema for structural models of object-oriented software systems*

The models distinguish four types of software entities: packages, classes, methods, and attributes. This means that they abstract from the detailed source code within the methods, which greatly enhances the scalability to large systems, and still enables good traceability of the visualization to the source code.

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, we decided to collapse each class with their contained classes. We feel that 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.

Such models can be automatically extracted from the source code of object-oriented software systems. In our experiments, we used the tools SNiFF+ [Win] and Sotograph [Sof] to extract graph models from Java programs.

The extracted graphs are stored in files in Rigi Standard Format ([Won98], Section 4.7.1), a simple tuple notation used by many reverse engineering and reengineering tools. Through the separation of extraction from visualization, and the use of a standard exchange format for graphs, Software Landscapes can visualize data from many sources. In particular, they can be applied to software in any programming language for which an appropriate extractor is available.

In this paper, we present visualizations of four software systems, all developed using the programming language Java: Eclipse 2.02 (an open source software development platform), JWAM 1.6 excluding test classes (a framework for interactive business applications), a commercial application (called SystemX for anonymity), and CrocoCosmos (a research tool for software analysis and visualization). Table 1 shows their characteristics, where LOC (lines of code) is the total number of carriage returns in the source code.

System	Entities	Relations	LOC
Eclipse 2.02	112 613	339 161	1 181 270
JWAM 1.6	11 097	25 081	167 178
SystemX	8 042	19 378	78 220
CrocoCosmos	1 269	2 574	16 832

Table 1: *Visualized software systems*

4. The Landscape Metaphor

The challenge in the visualization of complex software systems lies in the undistorted and comprehensible representation of the large amount of extracted data. This implies two goals: First, the information density in the views should be maximized under the constraint of comprehensibility. However, even with a large information density, not all details of the data can be represented in one view. Thus the second goal is to provide intuitive navigation, which allows the user to move easily between views on different levels of abstraction and of different parts of the visualized system.

Clearly, there is a tradeoff between information density and comprehensibility. Many three-dimensional visualizations have a high information density, but appear cluttered, occlude distant information, and provide no global overview. On the other hand, many two-dimensional visualizations are very clear but reveal little information. Therefore it is desirable to find a compromise that combines information density and clarity. A promising approach is the use of virtual landscapes, where three-dimensional objects are arranged on a two-dimensional plane.

These landscapes are similar to our physical environment, namely the (roughly two-dimensional) surface of the earth with three-dimensional buildings, trees, et cetera. This similarity allows us to transfer perceptual abilities acquired in earlier situations to the comprehension of and orientation in the visualization, thus reduces the load on conscious information processing.

A principal characteristic of the landscape metaphor is the hierarchy of different abstraction levels. For example, the world can be divided into continents, the continents into states, the states again into cities, which in turn consist of urban districts with many houses. The hierarchical structure of object-oriented software is naturally mapped to landscapes, and thus can be represented comprehensibly and with minimal distortion. Because we are familiar with the different levels of abstraction, we can easily navigate between them.

The term landscape metaphor is not to be understood exclusively as a detailed image of reality, but rather stands for structures that are similar to those of a real landscape. The concrete representations of the information in the form of objects of the landscape can possess both, a close-to-reality and an abstract appearance, e.g. as in maps. The pros and cons of each deviation from real landscapes must be evaluated carefully. While such deviations potentially confuse the viewer and lead to misinterpretations, they must be considered whenever important characteristics of the data cannot be clearly visualized with realistic objects.

In conclusion, the landscape metaphor offers a good tradeoff between information density and comprehensibility, it is familiar to us, and its structure is similar to the structure of software systems. However, we will accept deviations from real landscapes when this clarifies the visualization.

5. Software Landscapes

This section presents our approach for visualizing the structure of large software systems. The first subsection introduces methods to generate layouts according to the hierarchy of a software system. The visual representation of software entities and a level of detail mechanism are addressed in Subsection 5.2. Finally, the representation of the relations between entities is discussed in Subsection 5.3.

5.1. Hierarchy Based Layout of Entities

With the visualization of software structures, special attention is drawn to the production of expressive object arrangements. 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 spheres. The outermost sphere stands for the root of the package hierarchy. It contains spheres which represent the packages that are directly contained in the root package. These spheres for the second level packages again contain spheres for the third level packages, and so on. The spheres contained in a package are arranged on a circle within a two-dimensional plane. The size of the

spheres is adjusted to the size of the available segments of the circle, so that on one hand, they do not overlap, and on the other hand they do not form large gaps. Figure 2 illustrates the result of this arrangement pattern.

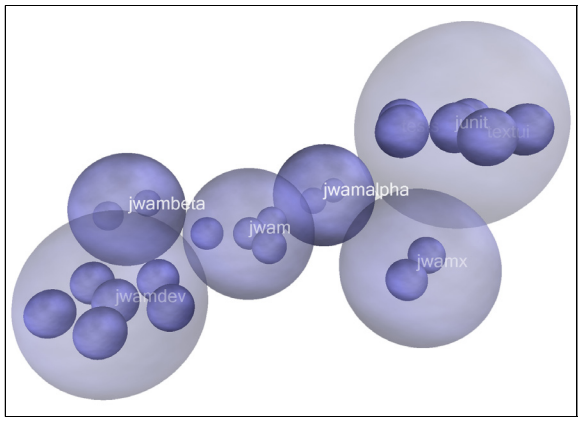

Figure 2: *Representation of the package hierarchy of the software system 'JWAM 1.6' using nested spheres*

After arranging the packages, the positions of the classes are defined. Therefore a platform is inserted into each sphere on which the classes are arranged. Every platform represents its own unique landscape. If a package does not contain classes, the representation of the platform in the respective sphere is discarded. Each class is represented as a circle with a surface area related to the number of contained methods and attributes. The circles are distributed on the platform with the help of the relaxation of Voronoi diagrams [HKL*99, DHvS00]. This method is also used to arrange the method and attribute objects on the class circles.

5.2. Visual Representation of Entities

As already explained in Section 5.1, nested spheres are used for the arrangement of packages. Pursuant to [RG93] the representation of the spheres 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 spheres 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 [RG93] 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 sphere is more than five times the sphere's radius, then the sphere is drawn perfectly opaque. If the distance of the viewpoint is less than two times the sphere's radius, then the sphere is completely transparent. Between these two distances, infinitely variable cross fading takes place (see Figure 3). On account of fading out distant levels it is possible 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 spheres does not have to be drawn, and normally more than 90 percent of all spheres are completely opaque.

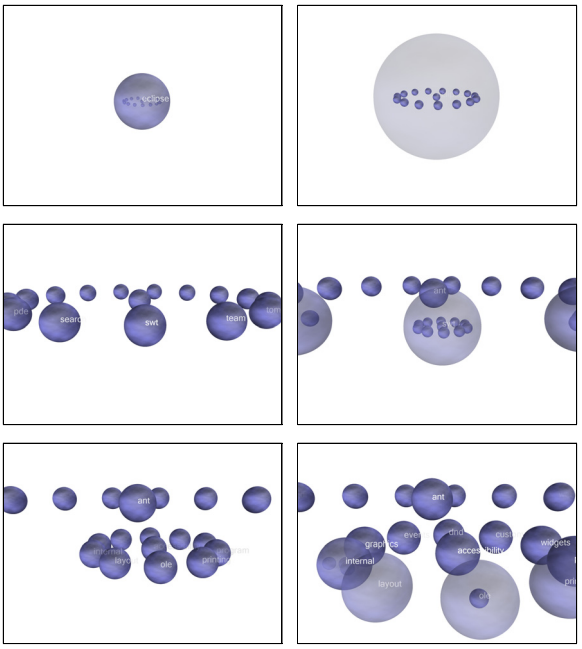

Figure 3: *Changing sphere transparencies when zooming into the package hierarchy of the software system 'Eclipse 2.02'; Far upper left: Overview of the entire software system; Far lower right: Closeup view of selected packages of the software system*

Circular discs are used for the representation of classes. Methods and attributes are positioned on these discs in the form of cuboids. The objects typifying the methods are larger and have a different coloration than the attribute objects. Figure 4 shows a resulting visualization.

5.3. Visual Representation of Relations

Aside from the hierarchy of the entities, the relations between the entities are an important part of the structure of software systems. Our models distinguish three types of relations: inheritance between classes, method calls, and accesses on attributes. 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 not possible. In the realm of this

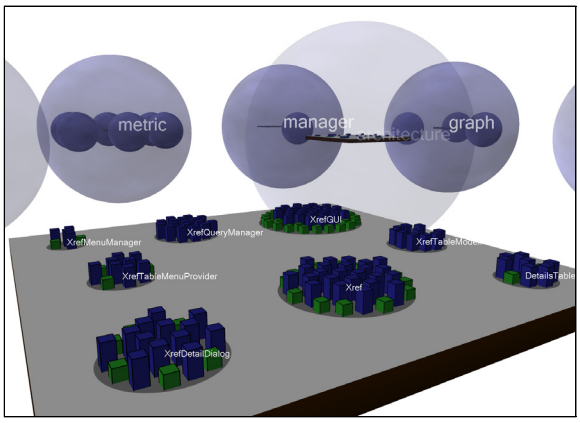

Figure 4: *Closeup view of the software system 'SystemX' containing 52 packages, 546 classes, 4856 methods and 2588 attributes*

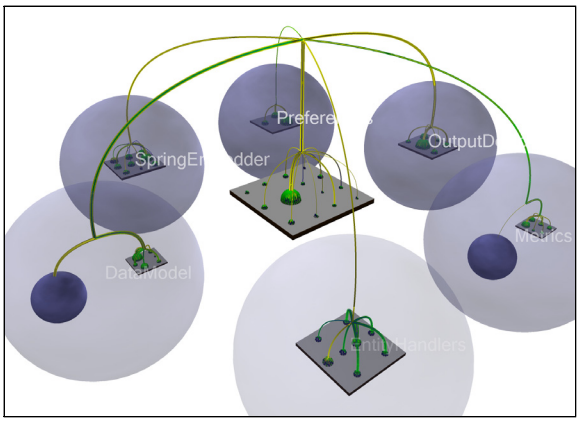

Figure 5: *Visualization of 2574 relations between 1269 entities within the software system 'CrocoCosmos' using a Hierarchical Net*

work, it was first attempted to solve the problem in the two-dimensional space. Based on diverse criteria the relations were summarized, occlusions were avoided and overlappings were reduced. The results for a few objects and relations were satisfactory, but a large number of participant entities and relations resulted in unsatisfactory representations. The conclusion was that for a clear representation of the relations, the third dimension has to be used.

We propose a solution called Hierarchical Net. Thereby the relations are represented not as direct connections between the involved objects, but 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 5.

The type and direction of the relations is shown by the color and brightness gradient of the corresponding lines. Connections of the same type and direction, and with the same start and end points, are combined to one connection. Thereby their quantity is mapped to the width of the new line, so that thicker lines stand for a larger quantity of represented connections. Because only lines of the same type and direction are combined, occlusion still appears, making thinner lines sometimes hard to recognize. In order to avoid this, lines are sorted by width and rendered in descending sequence. Thus it is possible to differentiate between the different line types and line directions despite mutual covering.

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.

6. Conclusions and Future Work

This work only represents a first step for the visualization of the structure of large software systems with the landscape metaphor. It introduced a layout technique for containment hierarchies, dynamic transparencies to reduce the visual complexity, and Hierarchical Nets to clearly represent the relations in large software systems.

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 the values of software metrics on landscape objects. For example, the height of the objects that represent methods could be proportional to the size of the methods, as measured by the number of lines of code.

References

[Cha93] CHALMERS M.: Using a landscape metaphor to represent a corpus of documents. In *Proceedings of the International Conference on Spatial Information Theory (COSIT)* (1993), LNCS 716, Springer-Verlag, pp. 377–390.

[CKTM02] CHARTERS S., KNIGHT C., THOMAS N., MUNRO M.: Visualisation for informed decision making; from code to components. In *Proceedings of the 14th International Conference on Software Engineering and Knowledge Engineering (SEKE)* (2002), ACM, pp. 765–772.

[CP01] CHEN C., PAUL R.: Visualizing a knowledge domain's intellectual structure. *IEEE Computer 34*, 3 (2001), 65–71.

[DHJ*98] DAVIDSON G., HENDRICKSON B., JOHNSON D., MEYERS C., WYLIE B.: Knowledge mining with VxInsight: Discovery through interaction. *Journal of Intelligent Information Systems 11*, 3 (1998), 259–285.

[DHvS00] DEUSSEN O., HILLER S., VAN OVERVELD C., STROTHOTTE T.: Floating points: A method for computing stipple drawings. *Computer Graphics Forum 19*, 3 (2000), 40–51.

[FdJ98] FEIJS L., DE JONG R.: 3d visualization of software architectures. *Communications of the ACM 41*, 12 (1998), 73–78.

[FHK*97] FINNIGAN P., HOLT R., KALAS I., KERR S., KONTOGIANNIS K., MÜLLER H., MYLOPOULOS J., PERELGUT S., STANLEY M., WONG K.: The Software Bookshelf. *IBM Systems Journal 36*, 4 (1997), 564–593.

[HDWB95] HENDLEY R., DREW N., WOOD A., BEALE R.: Narcissus: Visualizing information. In *Proceedings of International Symposium on Information Visualization* (1995), pp. 90–96.

[HKL*99] HOFF K., KEYSER J., LIN M., MANOCHA D., CULVER T.: Fast computation of generalized Voronoi diagrams using graphics hardware. In *Proceedings of the 26th Annual Conference on Computer Graphics (SIGGRAPH)* (1999), ACM, pp. 277–286.

[KM00] KNIGHT C., MUNRO M.: Virtual but visible software. In *Proceedings of the International Conference on Information Visualisation (IV)* (2000), IEEE Computer Society, pp. 198–205.

[Koi92] KOIKE H.: An application of three-dimensional visualization to object-oriented programming. In *Proceedings of the Workshop on Advanced Visual Interfaces (AVI)* (1992), World Scientific, pp. 180–192.

[LN03] LEWERENTZ C., NOACK A.: CrocoCosmos – 3d visualization of large object-oriented programs. In *Graph Drawing Software*, Jünger M., Mutzel P., (Eds.). Springer-Verlag, 2003, pp. 279–297.

[MOTU93] MÜLLER H., ORGUN M., TILLEY S., UHL J.: A reverse engineering approach to subsystem structure identification. *Journal of Software Maintenance: Research and Practice 5*, 4 (1993), 181–204.

[Obj03] OBJECT MANAGEMENT GROUP INC.: OMG Unified Modeling Language Specification, 2003.

[PFW98] PARKER G., FRANCK G., WARE C.: Visualization of large nested graphs in 3d: Navigation and interaction. *Journal of Visual Languages and Computing 9*, 3 (1998), 299–317.

[Plo97] PLOIX D.: Observation de programmes par la combinaison d'analogies. In *Actes de la conférence Intelligence Artificielle et Complexité* (1997), pp. 150–156.

[Rei95] REISS S. P.: An engine for the 3d visualization of program information. *Journal of Visual Languages and Computing 6*, 3 (1995), 299–323.

[RG93] REKIMOTO J., GREEN M.: The Information Cube: Using transparency in 3d information visualization. In *Proceedings of the 3rd Annual Workshop Information Technologies & Systems (WITS)* (1993), pp. 125–132.

[SM95] STOREY M.-A., MÜLLER H.: Manipulating and documenting software structures using SHriMP views. In *Proceedings of the International Conference on Software Maintenance (ICSM)* (1995), IEEE Computer Society, pp. 275–284.

[Sof] SOFTWARE-TOMOGRAPHY GMBH: Sotograph. http://www.softwaretomography.com.

[Sof03] *Proceedings of the ACM Symposium on Software Visualization (SOFTVIS)*. ACM, 2003.

[Vis02] *Proceedings of the 1st International Workshop on Visualizing Software for Understanding and Analysis (VISSOFT)*. IEEE Computer Society, 2002.

[WC99] WISS U., CARR D.: An empirical study of task support in 3d information visualizations. In *Proceedings of the International Conference on Information Visualisation (IV)* (1999), IEEE Computer Society, pp. 392–399.

[WF96] WARE C., FRANCK G.: Evaluating stereo and motion cues for visualizing information nets in three dimensions. *ACM Transactions on Graphics 15*, 2 (1996), 121–140.

[Win] WIND RIVER SYSTEMS INC.: Sniff+. http://www.windriver.com/products/sniff_plus/.

[Won98] WONG K.: *Rigi User's Manual, Version 5.4.4*, 1998. http://ftp.rigi.csc.uvic.ca/pub/rigi/doc/.