VANESSA: Visualisation Abstraction NEtwork for Software Systems Analysis

Michael J. Pacione
Department of Computer and Information Sciences, University of Strathclyde,
Livingstone Tower, 26 Richmond Street, Glasgow, G1 1XH, UK
michael.pacione@cis.strath.ac.uk

Abstract

Software visualisation is a comprehension technique with a wide variety of applications in software maintenance. However, current software visualisation tools do not address the full range of software comprehension requirements. This tool demonstration presents VANESSA: Visualisation Abstraction NEtwork for Software Systems Analysis, which is intended to address the shortcomings of existing tools. VANESSA is based on the use of abstraction combined with structural and behavioural perspectives of software. The tool is described and example analyses are presented.

1. Introduction

This tool demonstration describes and demonstrates VANESSA: Visualisation Abstraction NEtwork for Software Systems Analysis. Software visualisation is the process of modelling software systems for comprehension [12]. The comprehension of software systems both during and after development is a crucial component of the software process [14]. The complex interactions inherent in the object-oriented paradigm make visualisation a particularly appropriate comprehension technique. Software visualisation is therefore a useful technique in object-oriented software maintenance. The large volume of information typically generated during visualisation necessitates tool support.

A recent study by the author revealed that current visualisation tools address only specific software comprehension and reverse engineering issues [8]. The results of this study suggest that in order to address the full range of software comprehension and reverse engineering tasks, it is necessary to support a combination of abstraction and structural and behavioural information.

Previous work by the author has proposed a model that implements this approach [9, 10, 11]. It is reasonable to expect that a tool implementing this model would be useful in real world software maintenance. The aim of this research is to improve the effectiveness of visualisation techniques for large-scale software understanding. The motivation for this work was the lack of a unified model for software visualisation that allows the analyst to move conveniently between abstraction levels. The motivation in building VANESSA was to demonstrate the feasibility of the model, and to allow a rigorous evaluation of the model using real software systems to be performed.

2. Tool description

2.1. The underlying model

VANESSA implements a two-dimensional model consisting of abstraction levels and facets. The first dimension of the model consists of a hierarchy of abstraction levels from microscopic (intra-class) to macroscopic (business level). This arrangement allows the analyst to explore the software system at the level(s) of abstraction appropriate to the comprehension task they are undertaking.

The second dimension of the model consists of a number of *facets* [5], each representing a fundamental aspect of the system, such as structure or behaviour. The use of interrelated facets allows the analyst to examine the structure or behaviour of the software system individually or in combination, allowing them to focus the visualisation on the information appropriate to their query. Each abstraction level of each facet is a *view*.

The advantages of the integrated approach to visualisation taken in this model are that it provides a unified model that addresses all software comprehension requirements, and fills in any gaps left by individual tools. The model defines the visualisation space of the full range of comprehension requirements, and enables reasoning about and navigation between visualisations. As an example, an instantiation of the behaviour hierarchy is illustrated graphically in Figure 1 for part of the JHotDraw object-oriented framework [6].

2.1.1. Abstraction relationships

Fundamental to the model is the definition of abstraction relationships between the model views. These abstractions allow software visualisations to be related and manipulated. Such an integrated arrangement is preferable to an ad hoc approach as it allows software visualisations to be reasoned about in a coordinated and

correct manner. This approach addresses directly the fundamental issue of having to make use of a number of the current software visualisation tools to address the full range of comprehension requirements as such tools focus only on specific sub-problems of comprehension.

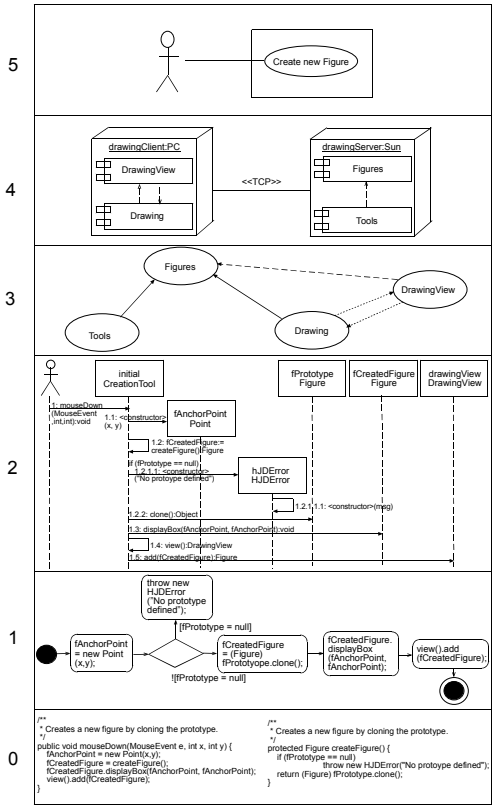

Figure 1. An example instantiation of the behaviour hierarchy

Abstraction operations are defined between each of the views in the model. These operations define the relative abstraction level of each view (i.e. that one view is more abstract than another), and define the transformations between views. Figure 2 presents the abstraction hierarchies implemented in VANESSA in the form of an abstraction network that illustrates these operations. The more abstract representation (the arc target) is derived from the less abstract base representation (the arc source) by applying the transformation indicated by the arc label. The three abstraction mechanisms used in the network are:

- abstraction by reduction (RED);
- abstraction by induction (IND); and
- partial systems morphism (PSM).

Fishwick [2] describes these mechanisms and presents examples in the context of a simulation of the dining philosophers problem.

In order to perform these abstractions, a set of mappings is defined between each view. When applying the mappings to a specific system, most mappings are generated automatically, while one or two may require knowledge of the system and its domain. In both hierarchies, the mappings between levels 0–1, 0–2, 1–2, and 3–4 are generated entirely automatically. The mappings between levels 2–3 can be generated automatically for some systems (e.g. based on source code naming conventions), but can also be produced using analyst knowledge of the system and domain. The mappings between levels 3–5 cannot be generated automatically and require input from the analyst.

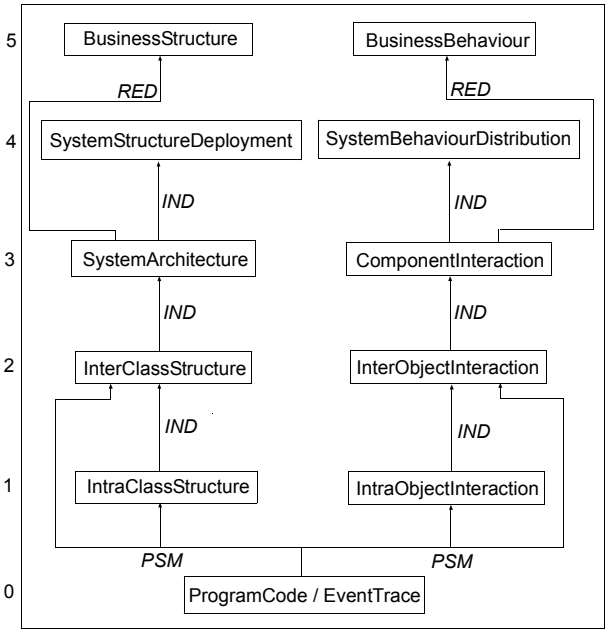

Figure 2. An abstraction network illustrating the relationships between the views

2.1.2. View combination

The rigorous definition of the abstraction mappings enables information from multiple views to be combined. Most CASE and software visualisation tools do not support this flexibility. View combination is useful for determining the low-level artefacts that correspond to high level system properties, and for focusing analyses. For example, an analyst may wish to examine the class structure responsible for some behaviour observed at the business level. The combination of level 2 structural and level 5 behavioural information would allow them to investigate this. The three possible scenarios for combining information are:

1. from the same level of each hierarchy;
2. from different levels of the same hierarchy; and
3. from different levels of each hierarchy.

View combination in VANESSA is demonstrated in Section 3.

2.2. Tool implementation

VANESSA is implemented in Java (J2SE 5.0) and analyses Java systems. Structural information is extracted statically from the program code. VANESSA first converts the source code to XML using BeautyJ [4], then manipulates the XML representation by means of an XSLT stylesheet using Xalan [1] to generate basic ('level 0') structure information. Behavioural information is extracted dynamically by generating an event trace. VANESSA incorporates a custom-built JPDA-based tracing utility implemented using JDI [13]. The trace generated contains the level 0 information for the behaviour hierarchy.

The next stage is to parse the generated level 0 information to produce level 1 and level 2 information. Once this process is completed, the abstraction mappings are applied to generate the higher-level views and the relationships between them. Expert mappings are read from text files if available. The ten views comprising the model hierarchies can now be output to files. The user can also specify any combination of views to be generated, as specified in Section 2.1.2. dot format [3] is currently used for output, though the generic nature of the model implementation allows any output format to be conveniently plugged in, such as UML. The output can now be viewed using a viewer such as dotty [7]. All aspects of the analysis occur in real time. The analysis process is illustrated in Figure 3.

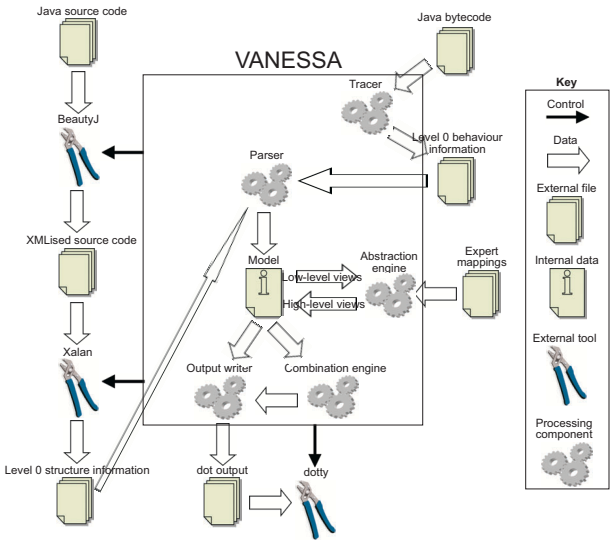

Figure 3. The VANESSA analysis process

3. Example analyses

This section demonstrates a number of example analyses of JHotDraw to illustrate the capabilities of VANESSA and the underlying model. Section 2.1.2 described the combination of views in the model. This section presents an example of each type of combination. This serves to illustrate both a selection of the basic views and the possible types of combination. In the following figures, less abstract entities are depicted nested within more abstract entities.

An example application of combining information from the same level of each hierarchy is to produce a unified visualisation of the structural and behavioural characteristics of the system, for example at the component level. This is accomplished in VANESSA by combining the level 3 views from the Structure and Behaviour hierarchies. Part of the result is illustrated in Figure 4.

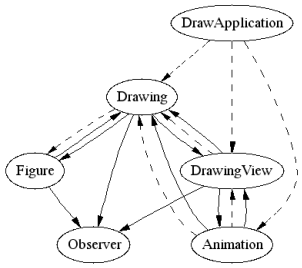

Figure 4. Combining views from the same level of each hierarchy. Solid arcs denote usage; dashed arcs denote dependency

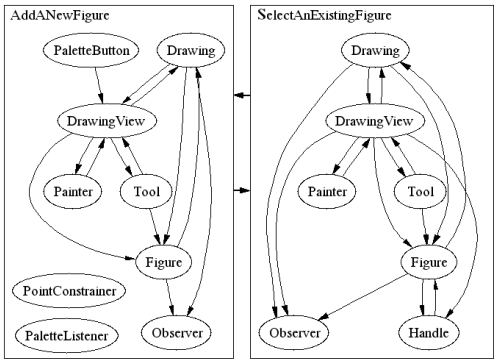

Figure 5. Combining views from different levels of the same hierarchy. Arcs between components denote usage; arcs between business entities denote business rules

An example application of combining information from different levels of the same hierarchy is to reveal the lower-level interactions responsible for the system's high-level behaviour, such as the inter-component interactions responsible for some business level behaviour. This is

accomplished in VANESSA by combining the level 3 and level 5 views from the Behaviour hierarchy. Part of the result is illustrated in Figure 5.

An example application of combining information from different levels of each hierarchy is to investigate the structural elements responsible for some high-level behaviour, such as the inter-class relationships responsible for some component level behaviour. This is accomplished in VANESSA by combining the level 2 view from the Structure hierarchy and the level 5 view from the Behaviour hierarchy. Part of the result is illustrated in Figure 6.

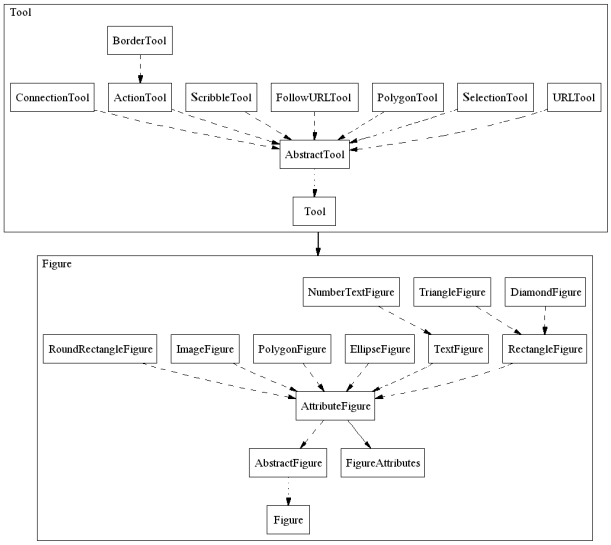

Figure 6. Combining views from different levels of each hierarchy. Between classes: solid arcs denote association; dashed arcs denote extension; dotted arcs denote inheritance. Between components: arcs denote usage

4. Summary

This tool demonstration has described VANESSA: Visualisation Abstraction NEtwork for Software Systems Analysis. This tool combines abstraction with structural and behavioural perspectives to address the full range of software comprehension requirements. Examples of the analyses possible with VANESSA were also presented.

VANESSA is currently a research prototype, and a number of enhancements remain to be implemented. In particular, it is planned to investigate how well VANESSA copes with larger systems, better interaction with the output, output to other formats such as UML, and combinations of more than two views.

VANESSA was created to demonstrate the feasibility of the visualisation model, and to allow a rigorous evaluation of the model to be performed. Having demonstrated the feasibility of the approach, the focus of future work will be on evaluating the effectiveness of the model in supporting software comprehension. The evaluation will employ a range of system types, and will make use of typical software comprehension tasks, as in previous work by the author [8, 10]. This will allow the effectiveness of the proposed visualisation model in supporting large-scale, real world software comprehension for maintenance to be assessed.

References

[1] Apache Software Foundation, *Xalan-Java*, http://xml.apache.org/xalan-j/, 2004.

[2] P.A. Fishwick, "The Role of Process Abstraction in Simulation", *IEEE Transactions on Systems, Man, and Cybernetics*, 18(1), IEEE CS Press, Los Alamitos, CA, 1988, pp. 18-39.

[3] E. Gansner, E. Koutsofios, and S. North, "Drawing Graphs with dot", http://www.graphviz.org/Documentation/dotguide.pdf , 2002.

[4] J. Gulden, *BeautyJ – Java Source Code Transformation Tool*, http://beautyj.berlios.de/, 2002.

[5] J.H. Jahnke, H.A. Müller, A. Walenstein, N. Mansurov, and K. Wong, "Fused Data-Centric Visualizations for Software Evolution Environments", *Proceedings of the 10th International Workshop on Program Comprehension*, IEEE CS Press, Los Alamitos, CA, 2002, pp. 187-196.

[6] R. E. Johnson, "Documenting Frameworks using Patterns", *Proceedings of the 7th Conference on Object-Oriented Programming, Systems, Languages, and Applications*, ACM Press, New York, NY, 1992, pp. 63-76.

[7] A. Koutsofios and S.C. North, "Editing Graphs with dotty", http://www.graphviz.org/Documentation/dottyguide.pdf, 1996.

[8] M.J. Pacione, M. Roper, and M. Wood, "A Comparative Evaluation of Dynamic Visualisation Tools", *Proceedings of the 10th Working Conference on Reverse Engineering*, IEEE CS Press, Los Alamitos, CA, 2003, pp. 80-89.

[9] M.J. Pacione, "Software Visualisation for Object-Oriented Program Comprehension", *Doctoral Symposium, Proceedings of the 26th International Conference on Software Engineering*, IEEE CS Press, Los Alamitos, CA, 2004, pp. 63-65.

[10] M.J. Pacione, M. Roper, and M. Wood, "A Novel Software Visualisation Model to Support Software Comprehension", *Proceedings of the 11th Working Conference on Reverse Engineering*, IEEE CS Press, Los Alamitos, CA, 2004, pp. 70-79.

[11] M.J. Pacione, M. Roper, and M. Wood, "Specification and Application of a Novel Software Visualisation Model to Support Software Comprehension", *Proceedings of the 21st International Conference on Software Maintenance*, IEEE CS Press, Los Alamitos, CA, 2005 (submitted for publication).

[12] B.A. Price, R.M. Baecker, and I.S. Small, "A Principled Taxonomy of Software Visualization", *Journal of Visual Languages and Computing*, 4(3), Elsevier, Amsterdam, 1993, pp. 211-266.

[13] Sun Microsystems, *Java Platform Debugger Architecture*, http://java.sun.com/j2se/1.5.0/docs/guide/jpda/, 2004.

[14] A. von Mayrhauser and A.M. Vans, "Program Comprehension During Software Maintenance and Evolution", *IEEE Computer*, 28(8), IEEE CS Press, Los Alamitos, CA, 1995, pp. 44-55.