A Proposal for Consistent Integration of Visual Computing with Visual Development of Software Components*

H. Ehrig
Technical University of Berlin

G. Taentzer
Technical University of Berlin

1 Summary

On one hand visual techniques are important for specification, modelling and programming, in short visual computing, and on the other hand they can be used to support component technology, evolution and the software development process, in short visual development of software components. The paper points out the central role of graphs and graph transformation for visual computing and visual development of software components. Moreover, it discusses the need for consistent integration of both areas and to propose graph transformation as a suitable technique for this purpose. The general idea of this position paper is not to present new technical results but to propose new lines of research how to combine and extend current projects and proposals in the areas of visual computing and visual development of software respectively.

2 The Role of Graphs and Graph Transformation in Visual Computing

Considering visual modelling languages which are one of the most investigated kind of visual languages (VLs), these languages are defined by meta models which are graphs (compare UML [13]). or visual grammars [8, 1]. A visual model such as an UML diagram, a Petri net, an SDL diagram, etc., can be either represented as an instance graph of a meta model or it

*Research partly supported by the TMR network GETGRATS, and the ESPRIT Basic Research Working Group APPLIGRAPH.

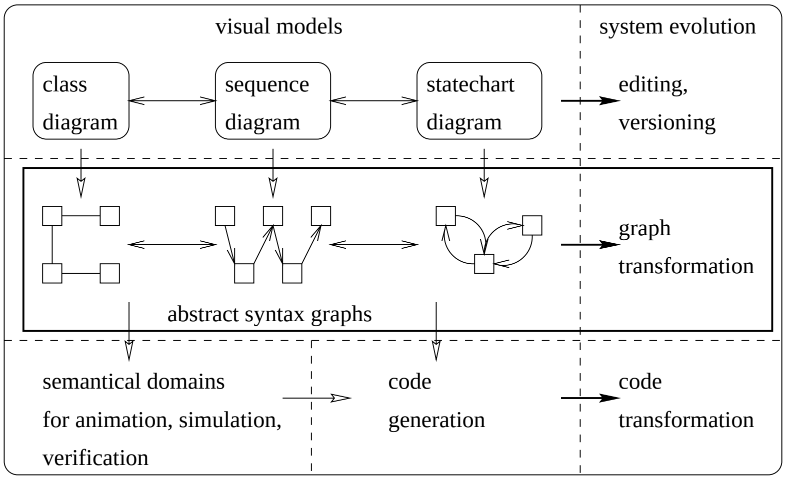

Figure 1: Visual development of software

can be generated by the corresponding visual grammar. No matter which sort of definition is used, graphs are natural to store such kind of visual information, i.e. n-dimensional structures.

Central for visual software development environments are abstract syntax graphs (ASG) [1] which are used to store the main information of visual structures (i.e. without concrete layout information). ASGs provided by a visual editor or lexical scanner can be used for various different tasks in the development environment (compare Fig. 1). For several of these tasks - concerning different properties of ASGs - it is quite natural to use graph transformation:

Parsing: Is the given ASG syntactically correct? Does it belong to a certain visual language? There are several approaches doing visual parsing by graph transformation as presented in [1].

Animation, simulation: If the ASG describes a dynamic structure, a certain control structure could be imposed on the ASG to perform the execution steps. It looks natural to define animation by transformation of an ASG extended like that.

Verification: Examining the ASG for further properties results in additional statements concerning the model or program, e.g. consistency checks (satisfaction of well-formedness rules in UML for each kind of modelling element) or conflicting transitions in Petri nets. Often the ASG has to be translated into a semantical domain to further inspect the model. This domain may be based on graphs again, e.g. reachability graphs for Petri nets to look for invariants. Consistency analysis of UML class and sequence diagrams us-

ing graph transformation techniques is studied in our DFG-project IOSIP[1] [11].
Code generation: Normally, the ASG is the source to generate a code skeleton or even executable code.
Data exchange: An input/output format relying on the meta model in a standardized way supports the exchange between various tools substantially.

Up to now, visual languages, especially the concrete visual layout, are subject of a fast development (e.g. class diagrams in the nineties), since there are much more possibilities to arrange language elements as for textual languages and these elements as well as their spatial relations are less standardized. Therefore, it is very convenient to use a generator for visual editors and other visual tools based on the definition of the underlying VL. It is a challenge to define a VL again visually, especially concerning the concrete layout. A promising approach is given by GenGEd [1] based on graph transformation and constraint satisfaction.

3 Visual Development of Software Components based on Graph Transformation

For the development of scalable, reusable software it is important to rely on a good notion of a component. Many visual languages suffer from the fact that they do not support components sufficiently. In general, a component provides a capsule around a piece of software with well-defined access points, i.e. import and export interfaces. An export interface contains services provided by the component whereas an import interface postulates the requirements for services of other components. Not all notions of components support a kind of import interface, e.g. the component diagrams in UML only know export interfaces. Components defined as graph transformation modules [5] or distributed graph transformation systems [12] support this more advanced interface concept.

Usually, there is a certain component view where the component structure of a system is shown (compare component diagrams in UML). Components supporting the advanced interface concept are shown in Fig. 2. This view does not contain details concerning the allocation of object and data structures, allocation of services and methods as well as signal and event handling. For these purposes the component view might be combined with further views showing different aspects of a system.

Performing visual computing of software componentwise means to handle several ASGs which may be interconnected by import/export parts.

[1] More information at: http://tfs.cs.tu-berlin.de/projekte/indspec/spp.

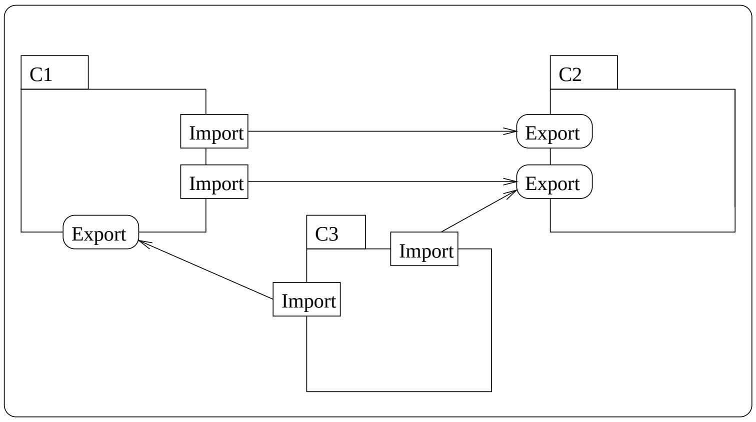

Figure 2: A component view

Since each component has its own ASGs for the different system aspects, it is possible to perform the development operations like editing, parsing, animating, verifying and code generating independently. But nevertheless, new sources of inconsistencies can occur, thus, the subject of consistent integration of components is as important as never before.

In contrast to components, views are always system parts which cannot live alone. Views can be defined along different aspects of a software system such as static structures, use cases and behaviour. On the other hand, views may be defined for different kinds of users and developers of software systems [2]. In both cases, views overlap in common modeling elements, programming constructs or human computer interface parts. These parts have to be kept consistent in different views during system development and evolution, e.g. several descriptions of one subject in different views have to be equivalent. Defining views and their interrelation by a distributed graph structure, advanced graph transformation can be used for inconsistency checks between several views [4].

Regarding Fig. 1 again, system development steps can occur on visual structures, on the internal ASG's, on translations in same semantical domain or directly on produced code. Typical development steps are editing, versioning, extension and redesign. For redesign graph transformation is used to propose a general proceeding for several tasks (e.g. transforming architectures [6], using evolution patterns [7], or distributing software [10]). Developing component-based systems, not the whole system has to be considered at once, but single components can be redesigned or replaced by others. A distributed structure of ASGs interconnected by interface parts can serve as the main base for consistent integration of several independent evolution steps.

4 Research Proposal for Extensions and Consistent Integration of Visual Concepts

In Sec. 2, we have pointed out the central role, graph transformation can play in visual computing. Some of these aspects are currently studied in the ESPRIT WG APPLIGRAPH, the TMR-projekt GETGRATS[2] and in the DFG-projects "Applied Graph Transformation" and IOSIP at TU Berlin. Moreover, this is a central issue of the new research training network proposal SEGRAVIS.

For visual development of software components based on graph transformation as discussed in Sec. 3 there are only very first approaches in the projects cited above. The central role of a component technology for software system development is one of the main requirements stated in the PITAC-report [9] concerning technical research priorities in the US. In Germany, component technology is the central issue in the BMBF-project KONTENG on "Continuous Engineering of Software and Information Infrastructures".

Although visual computing and visual development of software are important research fields in their own sight, it is clear that there is also a need for consistent integration of both areas, leading to a third important research field. Even without special focus on visual techniques it is considered in the KONTENG-project as an important problem to study consistency and integration of different components on one hand and of different views of one component (based on different specification techniques) on the other hand. Of course, it is even more difficult to handle consistency and integration of different components in different views. In the case of visual computing and visual development of software, we see a good chance to study these problems of consistency and integration: If both techniques are based on graph transformation - at least on the semantical levels as discussed in Sections 2 and 3 - we have already a common framework to study these problems.

The aspects of visual computing and visual development of software are also central issues of the new proposal of "Guaranteed Software Systems" currently in discussion by the EC for pro-active initiatives in the Future and Engineering Technologies (FET) program.

In view of these developments we strongly propose to extend the current SegraVis proposal in the sense of this position paper.

[2]More information to APPLIGRAPH and GETGRATS can be found at: http://www.informatik.uni-bremen.de/theorie//appligraph and http://www.di.unipi.it/ andrea/getgrats.

References

[1] R. Bardohl, G. Taentzer, M. Minas, and A. Schürr. Application of Graph Transformation to Visual Languages. In [3].

[2] G. Engels, H. Ehrig, R. Heckel, and G. Taentzer. A combined reference model- and view-based approach to system specification. Int. Journal of Software and Knowledge Engeneering, 7(4):457–477, 1997.

[3] H. Ehrig, G. Engels, H.-J. Kreowski, and G. Rozenberg, editors. Handbook of Graph Grammars and Computing by Graph Transformation, Volume 2: Applications, Languages and Tools. World Scientific, 1999.

[4] M. Goedicke, T. Meyer, and G. Taentzer. Viewpoint-oriented software development by distributed graph transformation: Towards a basis for living with inconsistencies. In Proc. 4th IEEE International Symposium on Requirements Engineering (RE'99), June 7-11, 1999, University of Limerick, Ireland. IEEE Computer Society, 1999. ISBN 0-7695-0188-5.

[5] Reiko Heckel, Hartmut Ehrig, Gregor Engels, and Gabriele Taentzer. A view-based approach to system modeling based on open graph transformation systems. In [3].

[6] J.-H. Jahnke and A. Zündorf. Rewriting poor design patterns by good design patterns. Technical Report TUV-1841-97-10, University of Vienna, 1997.

[7] M. Löwe. Evolution pattern. A Graphical Framework for Software Redesign. Proc. World Multiconf. on Systemics, Cybernetics and Informatics (1999), pp. 110-111.

[8] K. Marriott and B. Meyer. On the classification of visual languages by grammar hierarchies. Journal of Visual Languages and Computing, 8(4):375–402, 1997.

[9] National Coordination Office for Computing, Information and Coomunications. PITAC - Report to the President. Available at: http://www.hpcc.gov/ac/report, 1999.

[10] A. Radermacher. Support for design patterns through graph transformation tools. In Proc. 6th Int. Workshop on Theory and Application of Graph Transformation (TAGT'98), 1998.

[11] A. Tsiolakis and H. Ehrig. Consistency Analysis of UML Class and Sequence Diagrams using Attributed Graph Grammars. In H. Ehrig, G. Taentzer, editors, Proc. of Joint APPLIGRAPH/GETGRATS Workshop on Graph Transformation Systems, Berlin, March 2000, technical report no. 2000/2, Technical University of Berlin.

[12] G. Taentzer, I. Fischer, M Koch, and V. Volle. Visual design of distributed systems by graph transformation. In H. Ehrig, H.-J. Kreowski, U. Montanari, and G. Rozenberg, editors, Handbook of Graph Grammars and Computing by Graph Transformation, Volume 3: Concurrency, Parallelism, and Distribution. World Scientific, 1999.

[13] Rational Software Cooperation. Unified Modeling Language. Available at http://www.rational.com, 1999.

[14] B. Westfechtel. A graph-based system for managing configurations of engineering design documents. Int. Journal of Software Engineering and Knowledge Engineering, 6(4):549 – 583, 1996.

Hartmut Ehrig is with the Computer Science Department, Technical University of Berlin, Sekr. FR 6-1, Franklinstr. 28/29, D-10587 Berlin, Germany. E-mail: ehrig@cs.tu-berlin.de

Gabriele Taentzer is with the Computer Science Department, Technical University of Berlin, Sekr. FR 6-1, Franklinstr. 28/29, D-10587 Berlin, Germany. E-mail: gabi@cs.tu-berlin.de