Generative Software Visualization: Automatic Generation of User-Specific Visualizations

Richard Müller
rmueller@wifa.uni-leipzig.de

Pascal Kovacs
kovacs@wifa.uni-leipzig.de

Jan Schilbach
schilbach@wifa.uni-leipzig.de

Ulrich W. Eisenecker
eisenecker@wifa.uni-leipzig.de

Information Systems Institute
University of Leipzig
Leipzig, Germany

ABSTRACT

Software visualization provides tools and methods to create role- and task-specific views on software systems to enhance the development and maintenance process. However, the effort to produce customized and optimized visualizations is still high. Hence, we present our approach of combining the generative and the model driven paradigm and applying it to the field of software visualization. Based on this approach we want to implement a generator that allows to automatically generate software visualizations in 2d, 2.5d, 3d, or for virtual reality environments according to user-specific requirements.

Keywords

software visualization, model driven visualization, software visualization families, automation

1. INTRODUCTION

The preconditions for software visualization in 3d and virtual reality (VR) have improved dramatically, because of increased computing power available at low price and new presentation and interaction techniques. Our research tries to explore the resulting potential for software engineering, especially with respect to software development as well as maintenance.

Software visualization has the potential to considerably enhance understanding of software through providing structural, behavioral, evolutionary, or combined views [7]. This understanding is necessary for nearly all stakeholders involved in software development and maintenance, such as developers, project managers, and customers. All of these stakeholders have different tasks in different parts the of software lifecycle and therefore need different information about the software system they are involved in. Software visualizations have to support these users and their tasks, otherwise they are not useful and therefore will not be used. This task-oriented view was first proposed by Maletic et al. [12].

Reiss [16] identified some important issues of software visualization. Of these we address the lack of simplicity to use a visualization technique and the lack of adoption to real user problems. One reason for the lack of simplicity is, that visualization users need to supply exactly the data the visualization tool demands, because many tools require a special input format which the user has to provide. With our approach we are able to handle multiple software artifacts in multiple input formats, as long as they are in a well-structured format. The second issue, the lack of adoption to real user problems, comes with the mostly general scope of actual visualization tools. Reiss [16] takes the resource usage in the time around a specific event as an example for a real world use case, which is not covered by the available visualization tools. We believe that the adoption of principles, methodology, and techniques of software system families is the basis for developing a generator that adresses these problems. This generator takes one or more software artifacts and an easy-to-create configuration of the desired visualization as input. Furthermore, it provides ready-to-use visualizations optimized to the users requirements without the necessity of additional user intervention. These visualizations optimally support the different user needs and therefore the specific tasks in the process of the software development and maintenance. Moreover, the visualizations produced by the generator support any form of visualization technique be it two-, two-and-a-half- or three-dimensional (2d, 2.5d or 3d), printed on paper, displayed on a monitor, or presented in VR.

In this paper, we will describe the theoretical concepts of our approach and the design of the generator. In Section 2, we summarize theories and publications our work is based on. In Section 3, we explain the generative visualization process. In this context, we will first introduce the basics of Generative Programming, especially the generative domain model, and derive a generative software visualization domain model. Afterwards, we outline a technology projection to show how this model can be instantiated. To explain the characteristics of this approach, an example scenario demonstrates the process of generative software visualization in Section 4. The conclusion gives a brief evaluation of the work described in this article and provides an outlook to future research.

2. RELATED WORK

To respect the task-specific needs of different users, many tools for software visualization, e. g. Mondrian [14], CodeCity [17], or sv3d [13], allow to configure some aspects of the visualization. However, these tools are limited with respect

to their configuration options, e. g. the number of metaphors and layouts they offer. Furthermore, the configuration requires a substantial amount of additional manual work, or the tools are restricted to a single type of software artifact. Vizz3D [15] and Model Driven Visualization (MDV) [4] try to overcome these deficits with a more general approach.

Bull [4] describes MDV as an architecture for adapting the concepts of Model Driven Engineering (MDE). The software models used as input have to correspond to a platform independent metamodel, e. g. Dagstuhl Middle Model (DMM) [11]. Such a model can be retrieved by parsing sourcecode. To generate the visualizations, platform independent models called views are used, e. g. tree views or nested views. The definition of the necessary transformations between the input models and the view models have to be programmed by the user in a model transformation language, e. g. Atlas or Xtend[1]. After the transformation, a platform specific visualization will be automatically generated for a certain tool, e. g. Zest[2].

As a weakness of this approach, we identified the necessary creation of platform independent view models from scratch. On the one hand, this creates a high level of freedom. On the other hand, many possible benefits are prevented, such as using a common layout algorithm for different view models. Another drawback is the use of complex multi-purpose model transformation languages which are not easy to understand for non-experts. We show that using a formalized domain specific language (DSL) to describe the mapping from source to view elements will be easier to use while still preserving the automatic generation of a visualization.

Panas et al. [15] describe Vizz3D as a framework for configuring a visualization by using models and transformations. Beginning with a formalized model of software corresponding to a metamodel defined by Vizz3D, the user first configures the mapping to an abstract view. This view has a graph structure with nodes and edges including properties such as color or shape. In a second step, the user configures the mapping of the view to a concrete scene rendered by a tool, including the configuration of a metaphor and an optional layout.

A limitation of this approach is the required transformation of the users data into the Vizz3D source format, which causes additional effort. The user has to define two mappings to configure a visual representation instead of only one mapping. Finally this results in a tight coupling of platform independent metaphors to platform specific visualization tools.

3. GENERATIVE SOFTWARE VISUALIZATION

3.1 Generative Paradigm

Generative Programming aims at the automized production of software systems based on software system families:

> *"Generative Programming (GP) is a software engineering paradigm based on modeling software system families such that, given a particular requirements specification, a highly customized and optimized intermediate or end-product can be automatically manufactured on demand from elementary, reusable implementation components by means of configuration knowledge."* [6]

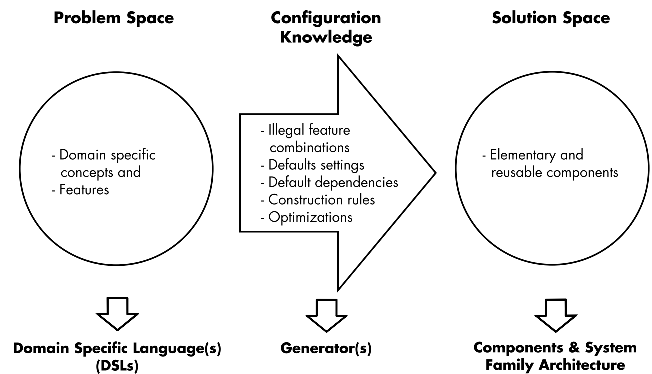

Figure 1: Generative Domain Model (GDM) [6]

Such a family covers a set of systems being similar enough from an architectural point of view to be built from a common set of assets. The requirements of the resulting system are described with a DSL. In this context, a domain is an area of knowledge comprising expert knowledge from stakeholders and technical knowledge of how to build software systems. A DSL is a specialized and problem-oriented language for domain experts to specify concrete members of a system family. It abstracts from technical knowledge and implementation details. This specification is processed by a generator, which automatically assembles the system by combining elementary and reusable components according to configuration knowledge and a flexible system family architecture as well. Components are building blocks for assembling different systems of a family.

The basic terms of the generative paradigm and their relationships are summarized by the generative domain model (GDM, see Fig. 1). It comprises the problem space, the solution space, as well as configuration knowledge for mapping the problem space to the solution space. The problem space covers domain specific concepts as well as their features used by domain experts to specify their requirements. The requirements are expressed in terms of one or more DSLs. The solution space includes elementary and reusable implementation components which can be assembled as defined through the system family architecture. The configuration knowledge encapsulates illegal feature combinations, default settings, construction rules, and optimizations as well as related information.

The parts of a GDM and two or more GDMs can be connected in different ways [5]. One possibility is that the solution space of one GDM is the problem space of another GDM. This is called a chaining of mappings. Furthermore, specifications in a DSL can be processed by different generators which map them to different solution spaces. In this case, there are several alternative solution spaces instead of only one.

3.2 Generative Software Visualization Domain Model

Comparing the generative paradigm with the field of software visualization, especially its visualization process, yields

[1]http://www.eclipse.org/modeling/emf/
[2]http://www.eclipse.org/gef/zest/

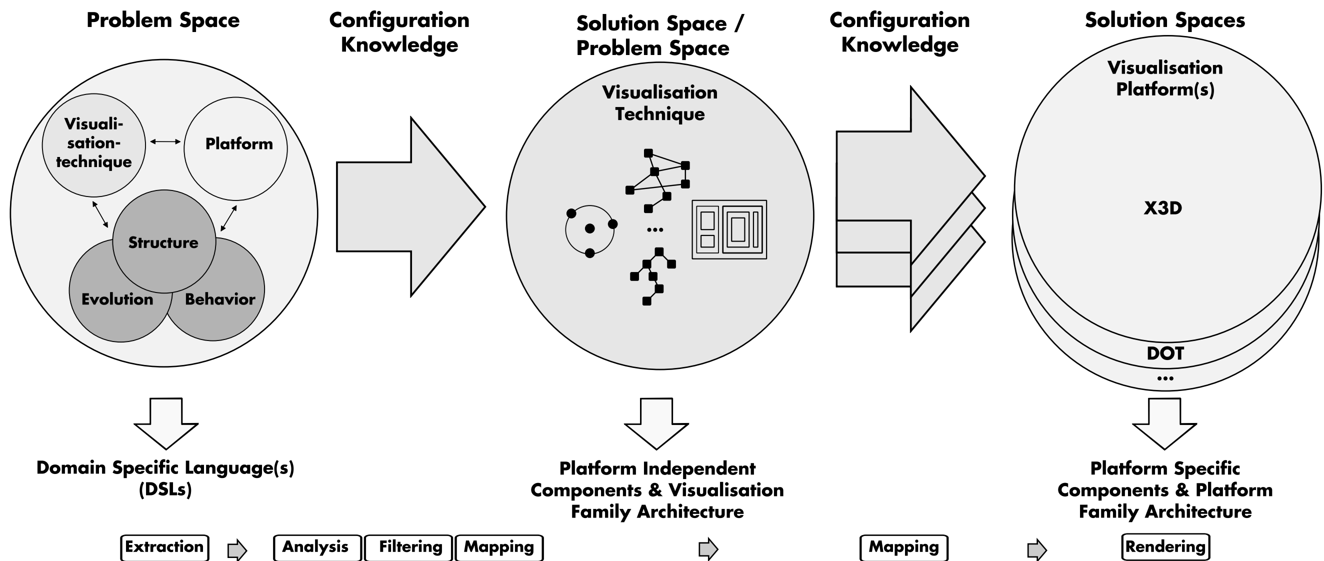

Figure 2: Generative Software Visualization Domain Model (GSVDM)

many remarkable similarities. A visualization should be automatically generated according to user-specific requirements by mapping information of software artifacts to a visual representation. For this reason, we adapt the definition of GP as follows:

> *"The visualization process should be arranged such that, given a particular requirements specification, a highly customized and optimized visualization can be automatically generated on demand from elementary, reusable implementation components belonging to a visualization family by means of configuration knowledge."*

The difference to the original definition is that the result of the generation process is not a software system but an optimized and ready-to-use visualization representing the structure, the behavior, and/or the evolution of a software system. Instead of assembling each visualization manually, it is created automatically from implementation components on the basis of a visualization family according to a specification in a DSL provided by the user.

This concept can be described in terms of the generative domain model. The chaining of mappings and the alternative solution spaces are used to realize the high variability of platforms for different visualization techniques. Besides that, the separation of the platform independent and the platform specific solution spaces makes it possible to reuse the implementation components. The resulting concept is called generative software visualization domain model (GSVDM, see Fig. 2).

The problem space offers means to specify concrete members of a visualization and a platform family for representing information from software artifacts. Using a DSL, the user can specify on which platform which information of a software artifact has to be visualized with which visualization technique. The DSL corresponds to the requirements specification in the above mentioned definition and abstracts from concrete implementations.

The solution space includes implementation components that can be assembled according to a family architecture. Due to the chaining of mappings, there are at least two solution spaces. The platform independent space contains abstract visualization techniques, such as trees, graphs, tables, and abstract or real world metaphors. The platform specific spaces provide concrete platforms for these techniques, such as Graphviz [10], Tulip [2], Gephi [1], or X3D [3].

The configuration knowledge, which is implemented as a generator, defines the mapping of problem space to solution space. In this case, the generation process is also the visualization process. This means that the process corresponds to a fully automated visualization process comprising all necessary parts of a visualization pipeline [8]. Thus, the knowledge about illegal feature combinations, default settings, default dependencies, construction rules, and optimizations is augmented with knowledge about extraction, analysis, filtering, mapping, and rendering from the DSL.

3.3 Model Driven Technology Projection

In order to implement this theoretical concept, it is necessary to identify concrete techniques for the elements of the software visualization domain model. Consequently, all spaces are described with structured models, and the configuration knowledge provides mappings between these spaces using model-to-model transformations. The DSL can be implemented as text-only or as a dialogue-based wizard controlling the different steps of the visualization pipeline. By using the XText-Framework[3] for implementing the DSL, we will be able to utilize existing functionality, to provide code completion, syntax highlighting and other useful features.

The starting point of the automatic visualization process are structured software artifacts containing information about structure, behavior, or evolution of software systems. Some of those structures – also known as metamodels – are Ecore, UML, XML or the DMM.

The common architecture for the visualization family is provided by a visualization technique meta-metamodel. It consists of a graph with nodes and edges where each element can have additional properties. The basic assumption

[3] http://www.eclipse.org/Xtext/

behind this is that all visualization techniques can be reduced to this abstract structure, so it is sufficient to have only one model that can be flexibly instantiated. Another advantage of using a graph structure is the ability to employ existing 2d and 3d layout algorithms rather than implementing them. The different platform independent visualization techniques are the implementation components.

It is obvious that the common architecture for the visualization platform depends on the used platform. For this reason, there is one model for each platform. Imagine the visualization should be an X3D-scene. Then, the X3D-file is the model, the X3D-schema definition is the metamodel, and the XML schema definition is the meta-metamodel. Further examples for visualization platforms are DOT[4] from Graphviz, TLP[5] from Tulip, or GEXF[6] from Gephi.

The configuration knowledge maps the elements of software artifacts to the elements of visualization techniques and finally to elements of a visualization platform corresponding to the DSL. The steps of the visualization pipeline are implemented by means of the Eclipse Modeling Framework. In this way, formal models can be analyzed and checked with predefined validation rules. For the mapping, i. e. model-to-model transformations, the transformation rules are defined on the meta-level and they are applied for each model conforming to the corresponding metamodel. In order to handle more than one source model the so called model-weaving is used. The rendering is done by the concrete visualization platforms.

4. EXAMPLE SCENARIO

In order to illustrate our approach we want to use a simple fictional scenario. Imagine a project manager who is preparing a meeting. In order to make the development team pay attention to current problems, information of the software system's structure enhanced with metrics is required. The system under development is a banking system implemented in Java and the relevant metrics are McCabe Complexity and LOC. To make it more understandable for all stakeholders, the manager waives source code and complex tables. Instead, the visualization should be 3d and represented by a nested visualization technique, which can be explored interactively in the company's virtual reality environment.

To do this in a generative way, the following steps have to be carried out. As a precondition, the necessary information has to be available in formal models, e.g. an Ecore model for the structure and an XML file for the metrics. Initially, the models are loaded by the generator (*extraction*). If necessary, the user can apply predefined rules to check the models for consistent semantics (*analysis*). Then, the relevant information from the software artifacts is selected (*filtering*), and a visual representation for each piece of information is defined by the user (*mapping*). This mapping comprises two stages: In the first stage, the visualization technique is selected, and in the second stage, the visualization platform is chosen. During these stages, the user sets the mapping rules for packages, classes, methods, attributes, and references to clusters, nodes, and edges as well as the visual appearance including shape, size, and colour of the different types of clusters, nodes, and edges. Clusters are special nodes, that can contain further clusters or nodes. The mappings are either completely controlled by the user or default mappings are applied. The simplified mapping rules in the banking example are as follows:

- package $\mapsto$ cluster $\mapsto$ brown cube
- class $\mapsto$ cluster $\mapsto$ blue sphere
- method $\mapsto$ node $\mapsto$ green to red cylinder
- attribute $\mapsto$ node $\mapsto$ blue cone
- reference $\mapsto$ edge $\mapsto$ blue line
- McCabe $\mapsto$ node $\mapsto$ green(low) - red(high)
- LOC $\mapsto$ cluster/node $\mapsto$ size

At this time, only the size and positions of the elements in the 3d space are missing. Here the graph structure from the meta-metamodel comes into play. By applying established layout algorithms the missing information is computed. For the banking example a force directed layout algorithm for clustered graphs is used [9]. Now, the generator has all necessary information to produce the visualization. As a result, the X3D-model in Fig. 3 and 4 is automatically generated

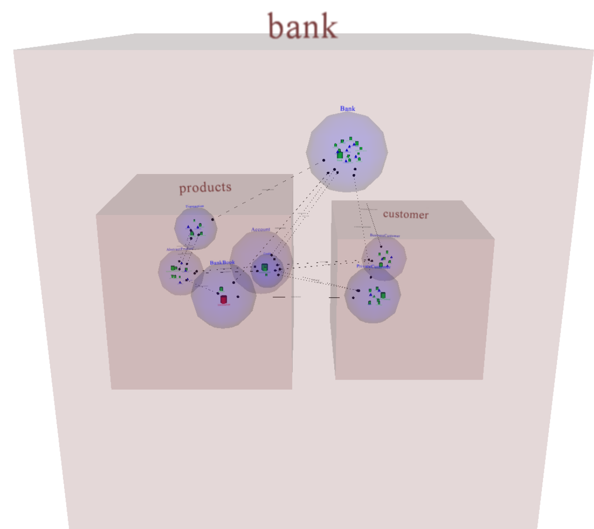

Figure 3: X3D-model of the banking example including structure and metrics (Overview)

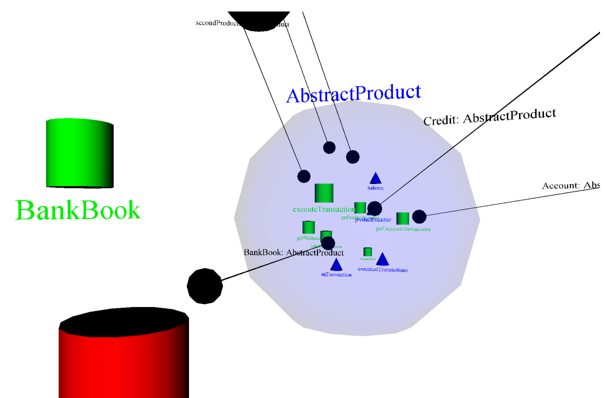

Figure 4: X3D-model of the banking example including structure and metrics (Detail)

[4]http://www.graphviz.org/content/dot-language
[5]http://tulip.labri.fr/TulipDrupal/?q=tlp-file-format
[6]http://gexf.net/

and can be interactively explored in a suitable browser, such as an Eclipse view, a standalone X3D-Browser, or a virtual reality environment. These visualizations have been generated using a predecessor of the planned generator. In this predecessor a box- and solar-system-metaphor as well as X3D as target platform are hard-wired. By this means, it was possible to visualize the structure of a real-world example with several hundred classes.

5. CONCLUSION AND FUTURE WORK

It was explained how the generative paradigm and the model driven paradigm can be adopted to meet the requirements of generating highly customized and ready-to-use software visualizations by the user without writing any glue code by hand. Hence, this promising concept makes it possible to integrate different kinds of software artifacts with different visualization techniques and well-approved visualization tools, not being limited to a specific platform and configured by an easy-to-use DSL.

Our future work will continue the implementation of the generator architecture and infrastructure based on the Eclipse platform, the specification of the grammar of the DSL, the iterative development of the meta-metamodel for visualization techniques including a representative amount of metamodels of visualization techniques as well as the integration of some established visualization tools. Eventually, we plan to use the generator to evaluate different visualization aspects, like three-dimensionality, animation, and interaction for their suitability in different tasks and different stages of the software life cycle. With the resulting findings we want to improve the spread of task- and role-specific software visualization in industrial software development and maintenance.

6. REFERENCES

[1] *Gephi: An Open Source Software for Exploring and Manipulating Networks*, 2009.

[2] D. Auber. Tulip : A huge graph visualisation framework. In P. Mutzel and M. Jünger, editors, *Graph Drawing Softwares*, Mathematics and Visualization, pages 105–126. Springer-Verlag, 2003.

[3] D. Brutzman and L. Daly. *X3D: Extensible 3D Graphics for Web Authors*. Elsevier, 2007.

[4] R. I. Bull. *Model Driven Visualization: Towards a Model Driven Engineering Approach for Information Visualization*. PhD thesis, University of Victoria, 2008.

[5] K. Czarnecki. Overview of generative software development. In *Unconventional Programming Paradigms*, number 3566 in Lecture Notes in Computer Science, pages 326–341. Springer, Berlin Heidelberg, 2005.

[6] K. Czarnecki and U. W. Eisenecker. *Generative Programming. Methods, Tools and Applications: Methods, Techniques and Applications.* Addison-Wesley Longman, Amsterdam, June 2000.

[7] S. Diehl. *Software Visualization - Visualizing the Structure, Behaviour, and Evolution of Software.* Springer, 2007.

[8] S. dos Santos and K. Brodlie. Gaining understanding of multivariate and multidimensional data through visualization. *Computers & Graphics*, 28(3):311–325, June 2004.

[9] T. Dwyer. Extending the WilmaScope 3D graph visualisation system: software demonstration. In *APVis '05: proceedings of the 2005 Asia-Pacific symposium on Information visualisation*, pages 39–45. Australian Computer Society, Inc., 2005.

[10] E. R. Gansner and S. C. North. An open graph visualization system and its applications to software engineering. *SOFTWARE - PRACTICE AND EXPERIENCE*, 30(11):1203–1233, 2000.

[11] T. C. Lethbridge, E. Plödereder, S. Tichelaar, C. Riva, P. Linos, and S. Marchenko. The dagstuhl middle model (DMM). http://www.site.uottawa.ca/~tcl/dmm/DMMDescriptionV0006.pdf, 2002.

[12] J. I. Maletic, A. Marcus, and M. L. Collard. A task oriented view of software visualization. In *VISSOFT 2: Proceedings of the 1st International Workshop on Visualizing Software for Understanding and Analysis*, pages 32–40. IEEE Computer Society, 2002.

[13] A. Marcus, L. Feng, and J. I. Maletic. Comprehension of software analysis data using 3D visualization. In *Proc. 11th IWPC*. IEEE Computer Society, 2003.

[14] M. Meyer, T. Gîrba, and M. Lungu. Mondrian: an agile information visualization framework. In *Proceedings of the 2006 ACM symposium on Software visualization*, SoftVis '06, pages 135–144, Brighton, United Kingdom, 2006. ACM. ACM ID: 1148513.

[15] T. Panas, R. Lincke, and W. Löwe. Online-configuration of software visualizations with vizz3d. In *Proc. 2nd SoftVis*, pages 173–182, New York, NY, USA, 2005. ACM.

[16] S. P. Reiss. The paradox of software visualization. In *Proc. 3rd VISSOFT*, pages 59–63, 2005.

[17] R. Wettel, M. Lanza, and R. Robbes. Software systems as cities: a controlled experiment. In *Proceeding of the 33rd international conference on Software engineering*, ICSE '11, pages 551–560, Waikiki, Honolulu, HI, USA, 2011. ACM. ACM ID: 1985868.