Towards a Compositional Approach to Define Graphical Animation of Software Applications *

Roswitha Bardohl
TU-Berlin, Germany
rosi@cs.tu-berlin.de

Leila Ribeiro
UFRGS, Porto Alegre, Brazil
leila@inf.ufrgs.br

Abstract

The PLATUS *simulation environment allows for a modular description of simulation models, clearly separating aspects of behavior, statistics and animation. Each component of a model in* PLATUS *has its own animation interface, that describes when and which messages will be sent to a corresponding animation module. The* GENGED *environment was originally developed for the visual definition of visual languages and the generation of language-specific graphical editors. Here we will extend* GENGED *in order to define several animation modules which can be connected via animation interfaces to* PLATUS *modules, allowing for a visual animation of the application being simulated.*

1. Introduction

A typical application where animation is needed is simulation. The main aims of simulating a system are to validate and recognize behavior and performance aspects. The first step to do a simulation is to build a model of the system to be simulated. Such models usually include descriptions of behavior, statistics gathering procedures and visualization (or animation) procedures. A visualization of the simulation (or animation of a model) may be quite important to validate the model, to have a visual feedback of the activities that are being executed in the simulation, and also may serve as a first version of an user interface for managing this system.

Typically, there are many different views of the simulation that may be shown in different animation windows at the same time. Moreover, to be really useful, an animation view shall be as close as possible to the visual language of the application domain being simulated. However, most of the existing simulation tools do not offer the possibility of constructing nice animation windows that may be reused for other simulations or even for the system under consideration. This has been the main motivation for our approach.

*Research partially supported by the German Research Council (DFG), and the projects PLATUS (CNPq and Fapergs) and GRAPHIT (CNPq and DLR).

2. Simulation and Animation

The PLATUS simulation environment [2, 3] is based on the formal calculus of graph transformation. Each component of a simulation model in PLATUS is represented by a graph grammar that is itself a composition of three graph grammars describing the behavior, the statistics and the animation. The statistics and animation grammars describe very abstractly when and which messages this component will send to a corresponding concrete statistics/animation module. These messages (together with the dependencies among them) are collected in statistics and animation interfaces which serve as abstract specifications of the statistics/animation expected for this component. Furthermore, the animation interface can be used as a basis to define several animation modules using GENGED [1].

Like PLATUS, also GENGED, an environment supporting the visual definition of visual languages (and corresponding editors), is based on graph transformation. We are going to extend GENGED in order to provide a user with a flexible way to define, use and change animation modules for the system being designed. Therefore we investigate the kind of composition needed, and how far the existing composition operators fulfill these needs. In particular, we investigate the kind of formal composition operator needed to allow a component-wise construction of the animation of a software application.

According to the PLATUS concepts, for each component of the simulation model, there may be several animation modules (a library), all of them being implementations of the animation interface of the component. Using GENGED, such an animation module is defined by an alphabet and a grammar, represented by a type graph and a graph grammar, respectively. For each simulation component needed in a certain application the user can choose an animation module from the library. As a simulation model is composed by various components, there may be also various animation modules that have to be composed accordingly. Once all animation modules are chosen and composed, the animation is triggered by corresponding messages sent from the simulation component to the application-specific animation

Proceedings of the IEEE Symposia on Human-Centric Computing Languages and Environments (HCC'01)
0-7695-0474-4/01 $17.00 © 2001 IEEE

environment during the simulation of the system.

In general, to build an animation view for an application composed by many different animation modules requires a way to suitably compose these animation modules into one view. As each animation is described by a graph grammar, we have to compose graph grammars. Moreover, this composition shall be semantics preserving, assuring that the animation of each component will be preserved when components are put together. An approach to composing graph grammars that preserves the semantics was presented in [7]. There, the composition was defined as a pullback construction (in a category of graph grammars). This construction was obtained component-wise by a gluing of type graphs (pushout in a category of graphs), a gluing of initial graphs (pushout in a category of typed graphs), and a combination of rule sets (pullback in a category of sets) where the rules in the composed grammar are obtained from the rules of the components by amalgamation, parallel composition or by just retyping rules of the component according to the type graph of the composed grammar. Our idea is to use this kind of composition to compose animation grammars. However, the definitions in [7] did not consider generalized graph structures nor attributes, they were defined for typed graphs [6]. Therefore we need to extend that definitions to consider the class of graphs we are using, that is, attributed graph structures.

We have done the first step: we analyzed the suitability of the main gluing construction of attributed graph structures (namely pushouts) to define the type and initial graphs of a composed animation grammar. Although this composition seems to be suitable, in general it is too restrictive: it would only allow to compose modules of attributed graphs having the same types (same graph structure signature) and attribute algebras to the same signature. Therefore, we need to consider more general categories of graph structures and algebras. This would lead to categories of generalized algebras [4]. Additionally, the considered attributed graphs may have different attribution functions, that is, the target category **SetP** shall be substituted by one that allows as morphisms families of functions indexed by different sets [1] .

3. Conclusion

The idea of combining PLATUS and GENGED is to have one common specification formalism to specify the behavior and animation of an application. In particular, with GENGED we expect to construct a library of animation modules that can be used to visualize the behavior of several software applications. We use graph grammars and graph transformation as the basis formalism.

The basic composition operator of attributed graphs (pushouts) is modeled in a way that it can be obtained component-wise. Moreover, there is a clear distinction beween the graph, the attribute algebra and the connection between them (separating abstract and concrete syntax, as required for the definition of visual languages [1]). Although the existing composition operator does not offer the power we need to compose modules, the modular structure of the definition (using generalized graph structures) gives us a good basis.

There are already many module concepts for graph transformation systems (graph grammars, however, without regarding initial graph), see [5] for a comparison. We hope to be able to use the main ideas of one of these concepts for adaptation to our composition operator (it is not possible to use standard composition operator for graph transformation systems to compose graph grammars because they are not compositional with respect to a graph grammar semantics). Another subject of future work is to allow different animation windows for the same application executed in parallel (one may want to visualize the same information in different ways).

[1] Due to the modular definition of attributed graphs used here, it will be possible to exchange the used categories by the general ones without losing the composition definition, as long as the general categories have some basic characteristics (like existence of pushouts).

References

[1] R. Bardohl. *Visual Definition of Visual Languages based on Algebraic Graph Transformation*. PhD thesis, TU Berlin, 1999.

[2] B. Copstein, M. da Costa M'ora, and L. Ribeiro. An Environment for Formal Modeling and Simulation of Control Systems. In *Proc. 33rd Annual Simulation Symposium*, pages 74–82. SCS, 2000.

[3] L. Ribeiro and B. Copstein. Compositional Construction of Simulation Models using Graph Grammars. In *Proc. Application of Graph Transformations with Industrial Relevance (AGTIVE'99)*, LNCS 1779, pages 87–94. Springer, 2000.

[4] H. Ehrig, M Baldamus, and F. Orejas. New Concepts for Amalgamation and Extension in the Framework of Specification Logics. In *Proc. ADT-Workshop*, LNCS 655, pages 199–221. Springer, 1991.

[5] H. Ehrig, G. Engels, R. Heckel, and G. Taentzer. Classification and Comparison of Modularity Concepts for Graph Transformation Systems. In 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, pages, 669–689, 1999.

[6] M. Korff. True Concurrency Semantics for Single-Pushout Graph Transformations with Applications to Actor Systems. In R. J. Wieringa and R. B. Feenstra, editors, *Information Systems - Correctness and Reusability*, pages 33–50. World Scientific, 1995.

[7] L. Ribeiro. Parallel Composition of Graph Grammars. *Applied Categorical Structures*, 7(4):405–430, 1999.