An OO Visual Language Definition Approach Supporting Multiple Views

D.H.Akehurst
University of Kent at Canterbury
D.H.Akehurst@ukc.ac.uk

Abstract

The formal approach to visual language definition is to use graph grammars and/or graph transformation techniques. These techniques focus on specifying the syntax and manipulation rules of the concrete representation. This paper presents a constraint and object-oriented approach to defining visual languages that uses UML and OCL as a definition language. Visual language definitions specify a mapping between concrete and abstract models of possible visual sentences, which can subsequently be used to determine if instances of each model "validly" express each other. This technique supports many:many mappings between concrete and abstract model instances, and supports the implementation of functionality that requires feedback from the abstract domain to the concrete.

1. Introduction and Background

Techniques for formally specifying textual languages have been in use for many years and the implementation of tools that make use of such languages is based on these formal techniques.

The formal approach to visual language (VL) specification is more complex. Graph Grammars (GG) are a technique used for specifying visual languages ([5]). The input concrete syntax is scanned to produce a Spatial Relationship Model (SRM); this is parsed using graph transformation ([1]) operations to construct the output Abstract Syntax Model (ASM).

The GG specification technique has three distinct drawbacks, as discussed in [5]:

- There is no clear distinction made between the two graphs, other than the names of the nodes;
- The combination of the two graphs and the added connecting arcs is cumbersome;
- The graphical part of the transformation rules is not expressive enough.

Our requirement is to build a tool that creates and updates the abstract model in parallel with the user's construction of the concrete expression. We also require a specification technique that keeps a clear distinction between the SRM and ASM.

The research behind this paper is an offshoot of a project ([8]) aimed at producing performance models from system designs. These requirements came from the need to capture the design of a distributed system, involving the specification of information from multiple viewpoints ([2]). Different VL's are appropriate for illustrating specifications in each viewpoint; each viewpoint may need multiple diagrams (views) to convey all the required information, but the underlying system model (ASM) is common across all diagrams and all viewpoints.

Graph translation is a process for specifying a mapping between two distinct graphs. The conventional approach is to combine the graphs by adding arcs between related nodes. A grammar is then defined over the combination. The technique for constructing one graph (the target) from the other (the source) is to alter the graph grammar into a set of rewrite rules that build the combined graph from parts of the source graph. The resulting target graph is consequently a subgraph of the combination.

The Triple Graph Grammars (TGG) approach ([7]) addresses the problems with GGs. Using this approach, separate grammars are produced for each of the graphs involved. The key is the production of a set of correspondence rules (the third grammar), which specify the homomorphic mapping between the two graphs.

An OO model can be considered as a graph of object-nodes and link-arcs. A GG could be defined to specify a pattern of possible models. However, a Class Diagram is an existing mechanism within the OO domain for specifying patterns of object models.

If an OO mechanism is developed for specifying a mapping between two class models, then we can achieve the same functionality as a TGG within the OO domain.

2. UML/OCL Specification Technique

The specification of a VL is considered to consist of five parts; three models and two sets of relationships. The models are the concrete syntax, SRM and ASM; the relationships define a mapping between the concrete syntax and SRM and between the SRM and ASM.

The concrete syntax to SRM mapping specifies the relationship between the concrete symbols and the more abstract SRM, which specifies the relationship between

the symbols. The SRM to ASM mapping is the equivalent of the parsing process, which constructs an abstract model from the "tokenised" (SRM) form of the concrete symbols.

Each of the three models is specified independently, and they are combined using the mappings to form VL-editor implementations. It is possible to define multiple concrete representations of the whole of or parts of an ASM, and/or use a CSM or its components to visualise multiple (independent) ASMs.

This VL-definition technique does not specify **how** to generate one graph (or model instance) from another, but it specifies if one model instance is a **valid expression** of the other. I.e. given a concrete model, an abstract model and a VL-definition, it should be possible to deduce if the models are valid interpretations of each other, according to the constraints of the given VL-definition.

The technique is best illustrated by following through an example. The visual language for Sequence Diagrams is used as the example. These diagrams have a complex structure and editors are not easily implemented.

[1] discusses the specification of the Sequence Diagram VL and illustrates the complexity and problems with the GG approach to the specification. The UML/OCL approach does not suffer from the same problems.

The symbols of a sequence diagram are essentially vertical lines (object lifelines) and horizontal arrows (messages). The connectivity of the symbols (i.e. the SRM) is similar to a directed graph. The arrows map to arcs in the graph, and the object lifelines map to nodes.

There is one major difference; the connectivity of the nodes to arc ends must be ordered, thus reflecting the order of the messages.

The three models are separately defined. The concrete symbols are defined in terms of appropriate rendering components. The SRM and ASM are defined as UML models. In this example, the ASM is the part of the UML meta-model that relates to interactions.

Mapping constraints are specified between components of each model, using UML concepts. An association is created for each mapping between classes. OCL constraints are specified in the context of this association. These specify constraints on instances of the mapped classes that must be met if the models as a whole are to be considered validly mapped.

The implementation of VL-editors based on this technique uses a two-way adaptation of the observer pattern. Each model is independently implemented as an OO data structure that fires events when it changes. The mapping objects are defined to listen to these events and make changes to the models that keep the constraints valid.

The concrete model can be implemented as a simple drawing editor. Events must be fired when symbols are added or removed, when their size or position changes or if the value of text components change.

The concrete to SRM mapping controls the visualisation of the connectivity modelled by the SRM. For example moving an object lifeline symbol must cause each arrow symbol that represents an arc connected to the moved lifeline's node, to be altered so that the connectivity remains correctly rendered.

The SRM and ASM models fire events indicating that value have been changed, or elements added or removed from collections.

3. Conclusion

A number of different VL's have been specified and implemented using this technique. These include class diagrams (including association class support), flowcharts and state machines. Work is in currently in progress for generating a Constraint Diagram ([3]) editor.

The technique enables reuse of much of the implemented code enabling rapid generation of new VL-editors.

The implementation structure allows multiple editors to be used to form expressions on the same underlying abstract model.

4. References

[1] Bardohl, R., Taentzer, G., Minas, M., Schürr, A.; Application of Graph Transformation to Visual Languages; *Handbook on Graph Grammars and Computing by Graph Transformation, Volume 2: Applications, Languages and Tools, World Scientific;* 1999.

[2] ISO/IEC; Open Distributed Processing - Reference Model - Part 1: Overview; *International Standard 10746-1, ITU Recommendation X.901;* July 1995.

[3] J Gil, J Howse, and S Kent; Constraint Diagrams: A Step Beyond UML; *Proceedings of TOOLS USA'99. IEEE Computer Society Press;* December 1999.

[4] M.Minas, G.Viehstaedt; DiaGen : A Generator for Diagram Editors Providing Direct Manipulation and Execution of Diagrams; *Proceedings of the 11th IEEE Symposium on Visual Languages;* September 1995.

[5] J. Rekers; On the use of Graph Grammars for defining the Syntax of Graphical Languages; *Proceedings of the colloquium on Graph Transformation;* 1994.

[6] J. Rekers, A. Schürr; A Graph Grammar Approach to Graphical Parsing; *Proc. Twente Workshop on Language Technology 10 joint with First AMAST Workshop on Language Processing;* 1995; pp. 163-172.

[7] A. Schürr; Specification of Graph Translators with Triple Graph Grammars; *Proc. WG'94 20th Int. Workshop on Graph-Theoretic Concepts in Computer Science, LNCS 903, Berlin: Springer Verlag;* June 1994; 151-163.

[8] Gill Waters, Peter Linington, David Akehurst, Peter Utton, Gino Martin; Permabase: Predicting the performance of distributed systems at the design stage; *To appear in IEE Procedings - Software;* 2000.