Automatically Generating Environments for Dynamic Diagram Languages

Mark Minas
IMMD II, Universität Erlangen-Nürnberg
Martensstr. 3, 91058 Erlangen, Germany
E-mail: minas@informatik.uni-erlangen.de

Abstract

This paper outlines a method to specify a wide range of diagrams by hypergraphs and diagram dynamics by hypergraph rewrite rules. The specification of a diagram language together with its dynamics can be used as input for an automated generator which creates a graphical environment for the specified dynamic diagram language. The environment serves as editor for diagrams of the specified language and as simulation system animating the execution of visual sentences.

1. Introduction

This paper describes continued work on DiaGen, a framework together with a generator for creating graphical editors for a specific diagram class from a formal specification [4, 6]. DiaGen has dealt with static diagrams or with limited dynamic extensions (e.g., finite automata) so far. Primarily, classes of graph-like and recursively built diagrams like Nassi-Shneiderman diagrams have been considered. This paper outlines that the concepts used in DiaGen—hypergraphs as an internal diagram model, hypergraph grammars, syntax-directed editing, and free hand editing of diagrams supported by hypergraph parsers—are also a simple means for specifying dynamic diagram languages which make use of general spatial relationships between their diagram components. For details of the following brief outline see [5].

There are several related approaches in the field of frameworks for VL environments based on diagrams; the most closely related ones are VLCC by Costagliola et al. [2] and the visual environment by Rekers and Schürr [7]. However, none of them allows for specifying diagram dynamics. VLCC uses an object-oriented hierarchy for representing diagrams according to their syntactic models instead of a uniform representation as in our work. For connecting visual components, VLCC uses attachment areas quite similar as described in the next section. Rekers and Schürr actually use two kinds of graphs as internal representations of diagrams: the spatial relationship graph (SRG) abstracts from the physical diagram layout and represents higher level spatial relationships. Additionally, an abstract syntax graph (ASG) is kept up-to-date with the SRG representing the diagram’s logical structure. Two different but connected graph grammars are used to define the syntax of SRGs and ASGs. DiaGen uses only one internal hypergraph instead of two graphs and requires only one hypergraph grammar which makes specification easier as we think.

For the dynamic aspects of VL environments, this work is closely related to work by Meyer and Marriott [3]. They propose a completely declarative approach which might be preferred, but we think that the operational approach outlined in this paper is easier to control.

2. Graph representation of diagrams

We use VEX [1] as a sample language of diagrams which primarily use spatial relationships for expressing λ-expressions: in VEX each variable identifier is represented by an empty circle that is connected by a line to a so-called root node. A root node is again an empty circle with one or more lines touching it, leading to all identifiers representing the same variable. A root node may either be internally tangential to another circle, it then represents a parameter of a λ-abstraction, or it is not included by any other circle, it then denotes a free variable. A circle representing a λ-abstraction contains its parameter circle and a VEX (sub-)diagram as its body. An application of two expressions is depicted by two externally tangential circles with an arrow at the tangent point. Figure 1a shows an example.

DiaGen uses hypergraphs as an internal diagram model: Each diagram class consists of a small set of diagram components, each offering a specific set of attachment areas. E.g., VEX has circles, lines, and arrows. A circle’s attachment areas are its borderline and its area. Lines and arrows have their end points as attachment areas. Each diagram component is represented by a hyperedge, nodes visited by the hyperedge represent its attachment areas. Figure 1b shows the (hyper)graph representation of the VEX diagram of Fig. 1a. Small circles represent nodes, ovals represent...
A hypergraph grammar is the appropriate specification technique for (2). When using free hand editing which allows to create also incorrect diagrams, a hypergraph parser has to be used to check whether a hypergraph is valid or not. For context-free and restricted context-sensitive hypergraph grammars, we have already presented efficient hypergraph parsers [4]. DiaGen now also uses a parsing procedure for more general hypergraph grammars. It is an extension of Zhang and Zhang’s parser which requires confluence of its transformation rules [8]. Confluence can be enforced by transformation rules which are extended by additional application conditions. For details of this parsing procedure see [5].

(3) By representing diagrams by hypergraphs, diagram dynamics (e.g., evaluation of VEX expressions) can be specified by hypergraph transformations. DiaGen uses hypergraph rewrite rules to define elementary modifications of the hypergraph, a specification of how to animate hypergraph transformations on the screen, and a control instance that controls application of transformation rules. For details see [5].

4. Conclusions

We have briefly outlined how hypergraphs can be used as a uniform means for modelling diagrams. When using free hand editing in a diagram editor, the hypergraph model is updated after each editing operation; a new hypergraph is easily modelled by adding resp. deleting corresponding hyperedges. Furthermore, the hypergraph can be used for specifying diagram dynamics by using hypergraph rewrite rules.

References