Chapter 8
Interactive Diagram Layout

Sonja Maier
Universität der Bundeswehr München, Germany

Mark Minas
Universität der Bundeswehr München, Germany

ABSTRACT
The specification of layout behavior for visual language editors is a challenging task. To support the user in an interactive environment, it is not sufficient to apply the same layout behavior in every situation. Instead, the user wants to select and alter the layout behavior at runtime.

The authors present a user study in which students created several visual language editors, mainly focusing on the layout engine. With the help of this user study, they demonstrate that different editors require similar layout behavior. They also show evidence that the combination of graph drawing algorithms and other layout algorithms is reasonable.

The authors also describe a framework that enables the creation of a layout engine, which satisfies the user demands identified in a user study. The approach is capable of defining layout behavior for various visual languages like graphs, class diagrams, mindmaps, business process models, or even GUI forms. The framework enables the combination of different layout approaches and provides the possibility to reuse already defined layout behavior.

INTRODUCTION
Layout is present in diverse areas, e.g., in text documents, webpages, or GUI forms. In this chapter, we focus on visual language editors (also called diagram editors), although our layout approach could be applied in many different domains. In Figure 1, a simple graph editor is shown, which will serve as the running example in this chapter.

In visual language editors, a layout engine is responsible for layout computation. This engine usually runs continuously and improves the layout in response to user interaction in real-time. Layout improvement includes all sorts of changes concerning the position or shape of diagram components. For instance, in case rectangle $E$ is...
moved (cf. Figure 1), the start point of the incoming edge is updated accordingly. In visual language editors, however, it is not reasonable to completely automate layout improvement. Instead, the editor users want to influence the layout at runtime. E.g., in Figure 1, the editor user has “defined” that nodes $D$, $H$, $I$, and $J$ are aligned horizontally, indicated by the two gray horizontal lines.

Two facts are highly relevant concerning layout in visual language editors: Users do not want to just visualize diagrams, but rather want to interact with them. They do not handle diagrams with more than 100 components, but usually diagrams with less than 100 components. Based on these facts, we can conclude that users favor predictable layouts that preserve the users’ mental map (Purchase, et al., 2007), instead of high quality layout derived from a standard layout algorithm, which aims at fulfilling some sort of graph aesthetics.

In general, we distinguish graph-like from non-graph-like visual languages. Many visual languages, however, fit into both categories. For instance, some parts of class diagrams show a graph-like structure, e.g., classes together with associations and generalizations. Other parts of class diagrams (c.f. Figure 4) show a non-graph-like structure, e.g., lists of attributes or the nesting of packages and classes.

In the context of visual language editors, the layout engine is usually implemented using one of the following approaches: hand-coded, graph