An Effective Layout Adaptation Technique for a Graphical Modeling Tool

Christian Seybold, Martin Glinz, Silvio Meier, Nancy Merlo-Schett
Institut für Informatik
Universität Zürich
CH-8057 Zurich, Switzerland
{seybold | glinz | smeier | schett}@ifi.unizh.ch

Abstract

Editing graphic models always entails layout problems. Inserting and deleting items requires tedious manual work for shifting existing items and rearranging the diagram layout. Hence, techniques that automatically expand a diagram when space is required for insertion and contract it when free space becomes avaliable are highly desirable.

Existing layout generation algorithms are no good solution for that problem: they may completely rearrange a diagram after an editing operation, while users want to preserve the overall visual appearance of a diagram.

We have developed a technique which automatically expands or contracts a diagram layout when items are inserted or removed while preserving its overall shape, i.e. the positions of the items relative to each other. Our technique has been implemented in a prototype tool. We are using it not just for simplifying editing, but primarily for implementing an aspect-oriented visualization concept.

1. Introduction

With conventional graphical modeling tools, editing a diagram always entails layout problems. When inserting an element into an existing diagram, the space required for the new element has to be created manually by shifting existing elements of the diagram apart. On the other hand, removing elements from a diagram yields empty space. Compacting the resulting diagram again has to be done manually.

Alternatively, automatic layout generation algorithms could be employed. However, applying such algorithms can result in a complete rearrangement of the diagram layout, thus destroying the so-called secondary notation, i.e. the positioning of the diagram elements relative to each other. As the secondary notation bears important clues for remembering and comprehending the meaning of a diagram, users typically want to preserve the secondary notation when modifying a diagram.

We have developed a layout adaptation technique which automatically expands or contracts a diagram layout when elements are inserted or removed while preserving the secondary notation.

Such a technique is not only convenient for inserting or deleting elements. It is a key prerequisite for implementing aspect-oriented visualization techniques. Assume a diagram that visualizes some basic elements of a model. When we want to display a particular aspect, the elements representing this aspect have to be woven (i.e. inserted) into the diagram. On the other hand, when hiding an aspect from a diagram, all elements belonging to this aspect have to be removed and the resulting diagram has to be compacted.

We use this adaptation technique in a tool for visualizing models written in ADORA [1]. ADORA is a modeling language which—in contrast to UML—integrates all modeling aspects (structure, behavior...) into one coherent model. To prevent the user being drowned in a flood of information, ADORA employs an aspect-oriented visualization concept: starting from a base view displaying a hierarchy of objects only, a structure aspect, a behavior aspect, etc. can be woven into the base view, resulting in a structure view, a behavior view, etc. Any combinations of aspects are possible.

2. An example

As an example, we describe the process of hiding aspects from an ADORA model of a heating control system. Fig. 1 shows a base view combined with the structure, behavior and user aspects. In Fig. 2, the latter two aspects have been hidden. Hiding these aspects means that the scenario *Manage Local Room Temperature* (represented as an oval) and all states and state transitions (represented as rectangles with rounded corners and as arrows, respectively) have to be removed from the diagram. Please note how the tool has automatically compacted the resulting diagram while preserving the secondary notation, i.e. the positions of the remaining objects relative to each other.

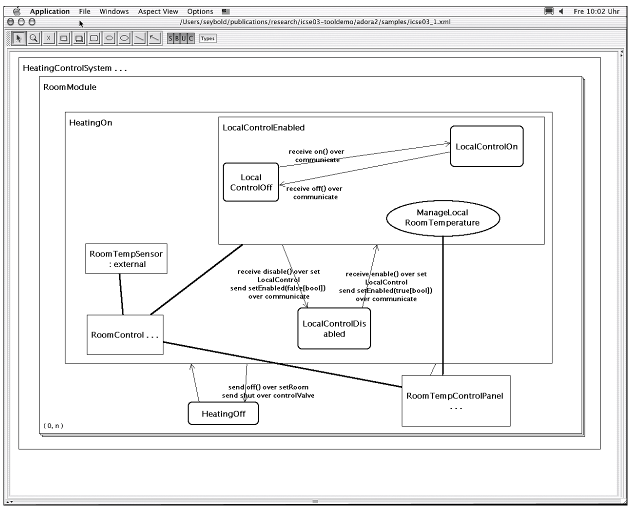
Application File Windows Aspect View Options
Fre 10:02 Uhr
/Users/seybold/publications/research/icse03-tooldemo/adora2/samples/icse03_1.xml
Types
HeatingControlSystem . . .
RoomModule
HeatingOn
LocalControlEnabled
LocalControlOn
receive on() over communicate
Local ControlOff
receive off() over communicate
ManageLocal RoomTemperature
RoomTempSensor : external
receive disable() over set LocalControl send setEnabled(false[bool]) over communicate
receive enable() over set LocalControl send setEnabled(true[bool]) over communicate
LocalControlDisabled
RoomControl . . .
send off() over setRoom send shut over controlValve
RoomTempControlPanel . . .
HeatingOff
(0, n)

Figure 1. **Base view woven with structure, user interaction and behavior aspects.**

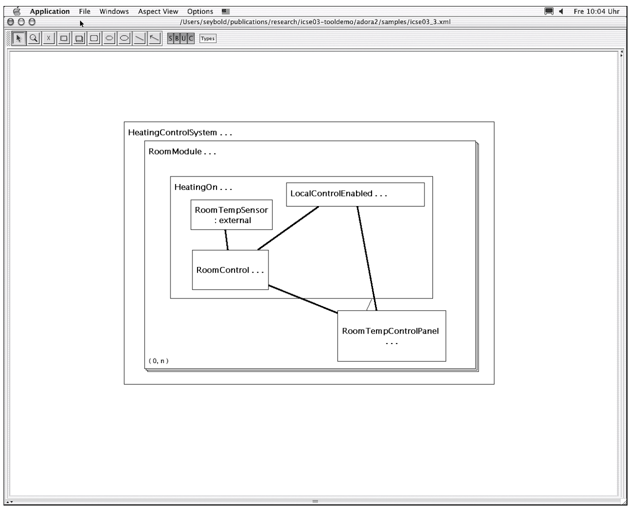
Application File Windows Aspect View Options
Fre 10:04 Uhr
/Users/seybold/publications/research/icse03-tooldemo/adora2/samples/icse03_3.xml
Types
HeatingControlSystem . . .
RoomModule . . .
HeatingOn . . .
LocalControlEnabled . . .
RoomTempSensor : external
RoomControl . . .
RoomTempControlPanel . . .
(0, n)

Figure 2. **Contracted diagram after hiding the user interaction and behavior aspects.**

3. Layout adaptation algorithm

In this section we sketch our algorithm. The details may be found in [4]. Below we describe the algorithm for compacting a diagram after removing items.

For simplicity, we consider removing a single object X and assume that all diagram items are rectangles. As we support a hierarchical modeling language, X is always embedded in a parent object P (root is of no interest here). Graphically spoken, P is a container for all its child objects, including X. Now let Z be the smallest bounding box containing all child objects of P before—and Z' after—X was removed. The idea is to keep the four horizontal and vertical distances from Z and Z' to the corresponding sides of P invariant. The algorithm works in four basic steps:
1. Shrink object X concentrically to a rectangle of zero size.
2. Shift all objects within P radially towards the center about half the size X was. Stop shifting objects when a minimum distance between them is reached (thus we avoid touching or overlapping objects). Their new positions define Z'.
3. Adjust the shape of P to the difference between Z and Z'.
4. Recursively apply the first three steps as long as P changes and in turn is contained in another parent object.

For expanding a diagram prior to inserting an item into it, we apply these steps in the same order, but in the reverse way (expanding instead of shrinking). Obviously, we can generalize this algorithm such that it handles removing or inserting a set of items with arbitrary convex shapes.

4. Conclusions

Achievements. We have developed an algorithmic technique that automatically adapts the layout of a diagram when elements are inserted or removed while preserving the secondary notation of the diagram (i.e. the relative positions of the elements). Our technique makes diagram editing considerably more comfortable for the user. Furthermore, it makes aspect-oriented visualization feasible for interactive tools. We consider this to be our main achievement.

Related work. We are not aware of any other modeling tool using similar techniques. Work has been done by others on automatic layout generation under constraints, where constraints can be used to specify structure-preserving layout adaptation [2], or on preserving a generated structure when solving overlappings [3].

State of work. The work reported here is part of a more general effort for developing a novel visualization concept for hierarchically structured models. Previously, we have developed context-preserving zooming algorithms for such models. All algorithms have been implemented and tested in our prototype tool for ADORA. We plan to fine-tune the algorithms and to conduct a validation study.

References

[1] M. Glinz, S. Berner, and S. Joos. Object-oriented modeling with ADORA. *Information Systems*, 27(6):425–444, 2002.

[2] W. H. Graf and S. Neurohr. Constraint-based layout in visual program design. In *Proceedings of the 11th International IEEE Symposium on Visual Languages (VL'95)*, 1995.

[3] W. Lai and P. Eades. Removing edge-node intersections in drawings of graphs. *Information Processing Letters*, 81:105–110, 2002.

[4] M. Marty. Analyse und Erweiterung von Algorithmen zur logischen Navigation in ADORA-Modellen [Analysis and extension of algorithms for logical navigation in ADORA models (in German)]. Diploma Thesis, University of Zurich, 2002.