Automatic Layout in the Face of Unattached Comments

Christoph Daniel Schulze
Department of Computer Science
Christian-Albrechts-Universität zu Kiel
Kiel, Germany
Email: cds@informatik.uni-kiel.de

Reinhard von Hanxleden
Department of Computer Science
Christian-Albrechts-Universität zu Kiel
Kiel, Germany
Email: rvh@informatik.uni-kiel.de

***Abstract*—Visual languages based on node-link diagrams are widely used for systems modeling. As in textual languages, comments can make diagrams easier to understand. In the absence of an explicit attachment between comments and the diagram elements they relate to, that relationship is usually given implicitly by the manual placement of comments near the related elements. While algorithms for the automatic layout of diagrams can make working with diagrams more effective, they usually fail to preserve implicit attachments by placing comments at arbitrary positions.**

In this paper, we propose a comment attachment algorithm that extracts implicit attachments and makes them accessible to layout algorithms. We implemented the algorithm in an application for browsing Ptolemy diagrams and achieved success rates, i. e. attachments as intended by the user, of up to 90%.

I. Introduction

The automotive industry has widely adopted visual languages to model software systems. Languages such as Simulink (The MathWorks), SCADE (Esterel Technologies), ASCET (ETAS), or Ptolemy (UC Berkeley) allow developers to define complex software systems graphically through *node-link diagrams*: nodes (also called vertices) consume and produce data that is transferred between them through links (also called edges), as seen in Fig. 1. The general assumption is that diagrams are easier to read and understand than traditional textual languages. This is not automatically true, however. Petre for example argues that visual languages are not easier to understand simply because they are visual [1]. Just like in textual languages, one tool for helping developers produce diagrams that are easier to understand are comments.

In diagrams, comments are usually represented as a special kind of vertex that contains text entered by the developer. Some visual languages allow comments to be explicitly attached to the diagram elements they relate to, if any; such an explicit attachment is usually visualized as a direct line between comment and diagram element. Other languages do not provide such explicit attachments; instead, the developer has to rely on other cues. The diagram in Fig. 1 has very well placed comments, making this an easy task. Usually, though, figuring out attachments is more complex, and is generally more difficult than in textual languages, where comments directly precede or follow the code they relate to, without much ambiguity.

Since the readability of diagrams strongly depends on their layout, developers often spend a lot of time moving vertices around [2]. Automatic layout algorithms are a way to free developers from constantly having to rearrange diagram elements while editing a diagram, allowing them to focus on the semantics they want to express. If there are explicit attachments between comments and diagram elements, layout algorithms can easily place them in close proximity. In the absence of explicit attachments, however, most layout algorithms will place comments at arbitrary locations, destroying any implicit attachment cues in the process, perhaps even introducing misleading ones. What is required for layout algorithms to properly place comments, thus, is a way to infer implicit attachments and make use of them to place comments and attached vertices in close proximity.

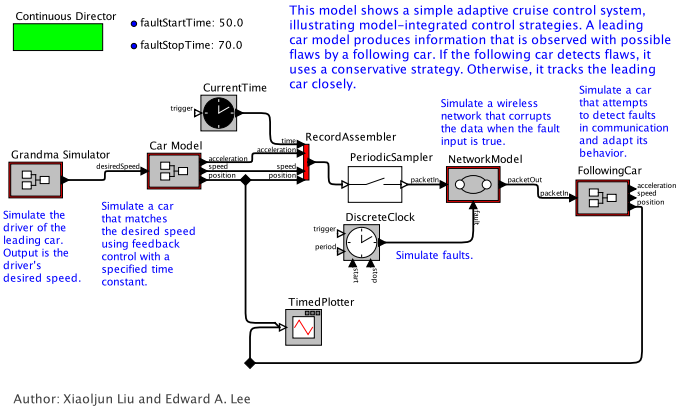
In this paper, we propose a simple distance-based algorithm for inferring attached vertices, explicitly excluding cases where comments may be attached to other elements of a graph, such as ports and edges. We evaluate the algorithm with a subset of the demo models that ship with Ptolemy.

We will start with a brief review of related work, followed by theoretical foundations. We then describe our algorithm, evaluate the results it produces and conclude the paper with a discussion.

A. Related Work

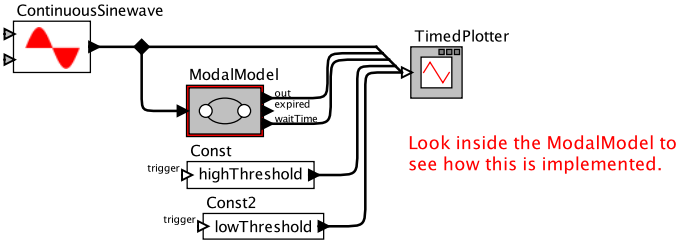
When it comes to automatic layout algorithms, Misue et al. distinguish between layout adjustment and layout creation [3]. Layout adjustment algorithms try to preserve a diagram's topology and will thus be better at preserving the visual cues present in the diagram. Layout creation algorithms compute new layouts from scratch and can easily remove the visual cues if no further effort is made. In this paper, we address layout creation algorithms.

To our knowledge, there has been no previous work on integrating comment positioning with layout creation algorithms. However, much research has centered around the related topic of integrating labels into layout algorithms. To name just two papers, Klau and Mutzel showed how to integrate edge labels into the orthogonal approach to graph drawing [4] and Kakoulis and Tollis proposed a method for placing node and edge labels in arbitrary drawings of graphs [5]. Contrary to comments, the attachment between labels and diagram elements is known in most graphical editing frameworks.



Author: Xiaoljun Liu and Edward A. Lee

Fig. 1. A sample diagram taken from the repository of demo models that ship with Ptolemy. Vertices consume and produce data that is transported through the edges. The comments describe either the model as a whole or individual vertices. Note how the placement of the comments makes attachments readily visible, even if the text may be too small to read here.



Authors: Edward A. Lee and Haiyang Zheng

Fig. 2. Part of another of Ptolemy's demo diagrams. Here, proximity alone does not suffice to infer the attachments. The comment to the right actually belongs to the vertex in the center, which becomes apparent only from reading the comment's text. The comment at the bottom has no relation to the vertex it is near to; it follows the Ptolemy convention of placing the names of the authors at the bottom left corner of the diagram.

II. Graphs and Annotations

We model diagrams as graphs and define a graph to be a pair $G = (V, E)$ of a finite set of vertices V and a set E of edges, either directed or undirected. For this paper, we augment this definition with a finite set C of comments. Vertices and comments have a width and a height as well as a position in the plane.

A *comment attachment algorithm* takes a graph with comments $G = (V, C, E)$ as its input, along with the size and coordinates of each vertex and comment. The aim is to compute a (possibly partial) comment attachment function $\mathrm{att}\colon C \to V$ that defines which vertex a comment relates to, if any. If a comment $c \in C$ is not attached to a vertex, $\mathrm{att}(c)$ is undefined and we write $\mathrm{att}(c) = \perp$.

III. Attaching Comments

When viewing a diagram, a human viewer will use several cues to determine the vertex a comment relates to. The easiest cue is an explicit attachment, shown for example as a direct link between comment and vertex in the diagram. Its advantage is clarity: an explicitly shown attachment makes the relation between comment and vertex unambiguously clear. In its absence we have to resort to other, less clear cues.

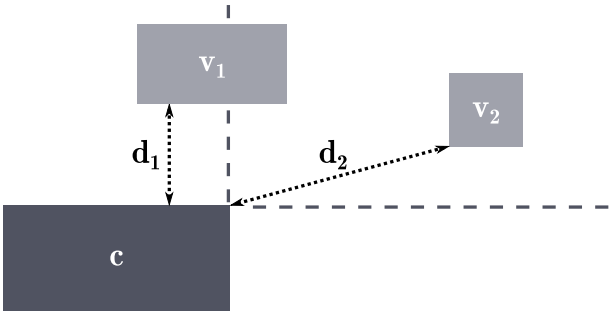


Fig. 3. Measuring the distances d_1 and d_2 between a comment c and two vertices v_1 and v_2.

The most obvious cue is proximity: the closer a comment is to a vertex, the more likely we will assume the two to be attached. This principle seems very basic, and one might be tempted to expect every developer to follow it instinctively. However, we have found examples of diagrams that violate this principle. Fig. 2 shows parts of a diagram whose author comment was placed in close proximity to a vertex it has no relation to. The reason for this can often be found in other design considerations that were deemed more important, for example to produce as compact a diagram as possible, or in explicit or implicit conventions of the modeling language. Whatever the cause, the implication is that proximity alone is not sufficient to infer implicit attachments.

Another cue is the content of a comment: if the comment mentions a specific vertex or gives enough information to constrain the vertices the comment can logically relate to, this will help to infer the correct attachment. A simple example of this is shown in Fig. 2, where the rightmost comment mentions a particular vertex.

A third cue are conventions of the visual language at hand: just like Java stipulates how method comments should be written, a visual language may stipulate where certain kinds of comments should be placed. For example, general diagram comments might always be placed at the top of the diagram, or the names of the authors might always be mentioned in the bottom left corner of the diagram (the diagram in Fig. 1 follows these conventions). Petre argues that knowing and making use of such conventions is what distinguishes expert programmers from novice ones [1].

The Algorithm

To achieve a near-perfect success rate, a comment attachment algorithm would have to make use of all of the cues mentioned above. However, some are harder to grasp algorithmically than others. Proximity is fairly easy to define and to compute, as are the conventions mentioned so far. Parsing the content of a comment, e.g. to retrieve vertex names, however, may be more complex.

Based on these considerations, we decided to have our algorithm rely only on proximity to infer attachments and evaluate if this already yields acceptable success rates.

To compute proximity, we need an exact definition. Let $G = (V, C, E)$ be a graph with comments. Let $c \in C$ and $v \in V$ be a comment and a vertex, respectively. We define

the proximity of c and v as the smallest distance between any two points on their border, as shown in Fig. 3. If v extends into the space above, below, left, or right of c, this is equal to the distance between the closest sides of the borders of c and v; otherwise, it is equal to the distance between the closest corners of c and v. In the algorithm and in the evaluation, we actually use the square of this distance to avoid having to compute square roots that we would need when computing the distance between corners.

For each unattached comment the algorithm now simply computes the vertex it is closest to. If the distance between the two does not exceed a certain threshold value, called the *maximum attachment distance*, the algorithm deems them to be attached; otherwise, the comment is not considered to be attached to any vertex. As the evaluation will show, finding a suitable maximum attachment distance greatly influences the success rate of the algorithm.

One detail that should be considered is whether to run the algorithm even if a diagram already contains at least one explicit attachment. In our view, the presence of explicit attachments suggests that unattached comments were left unattached for a reason. We therefore disable the algorithm if we find explicit attachments.

IV. Evaluation

We implemented the algorithm in the KIELER Ptolemy Browser,[1] an application for browsing Ptolemy models. To compute layouts, we use *KLay Layered*, a layout algorithm based on the layer-based approach to graph drawing introduced by Sugiyama et al. [6]. The algorithm was extended to place comments directly above or below the vertex they are attached to.

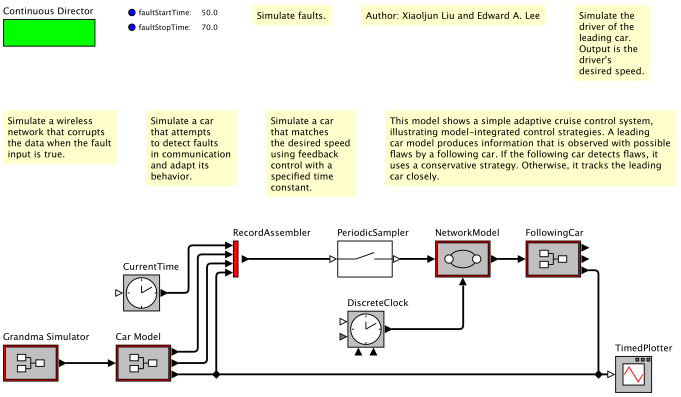
For the evaluation, we selected diagrams from the set of example models shipping with the Ptolemy tool that we were able to open properly in the KIELER Ptolemy Browser. Diagrams were selected if they contained at least one comment (which 148 diagrams did not), if they did not contain comments that relate to diagram elements other than vertices or that relate to more than one vertex (which 67 diagrams did), and if they did not contain attachments already explicitly defined by the user (which further 14 diagrams did). The remaining 308 diagrams had between 2 and 156 vertices (averaging 18.46) and between 0.02 and 1.5 comments per vertex (averaging 0.29).

We first determined the attachment of each comment for all diagrams manually, using all the clues mentioned in Sec. III, thereby defining a reference attachment function att_{ref}. To determine the quality of our algorithm, the attachment functions att_d, computed by our algorithm for various maximum attachment distances d, were compared to att_{ref}. For every comment $c \in C$, we distinguished four cases:

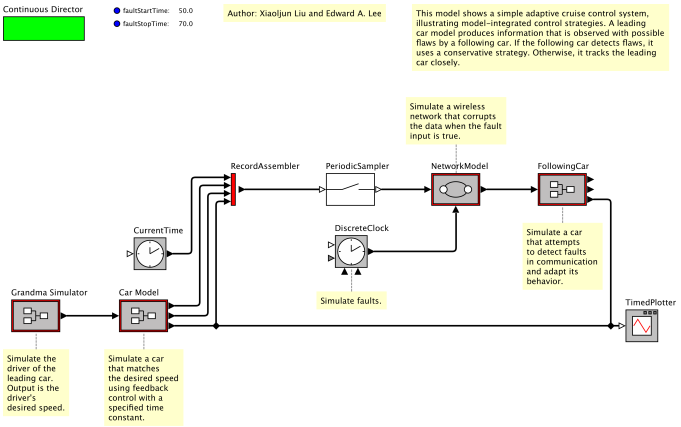
- *Correct attachments:* $\mathrm{att}_d(c) = \mathrm{att}_{ref}(c)$
- *Spurious attachments:* $\mathrm{att}_d(c) \neq \perp \wedge \mathrm{att}_{ref}(c) = \perp$
- *Lost attachments:* $\mathrm{att}_d(c) = \perp \wedge \mathrm{att}_{ref}(c) \neq \perp$
- *Changed attachments:* $\perp \neq \mathrm{att}_d(c) \neq \mathrm{att}_{ref}(c) \neq \perp$

We then counted how often each event occurred.

[1] http://informatik.uni-kiel.de/rtsys/kieler/



(a) If the comment attachment heuristic is switched off, the comments are placed arbitrarily, without any hint as to the vertex they relate to.



(b) If the heuristic is switched on, the layout algorithm can place comments and vertices in close proximity. Even if the text is too small to read here, the attachments are readily visible.

Fig. 4. The diagram from Fig. 1 as drawn by our Ptolemy browser, with a layout computed by our automatic layout algorithm.

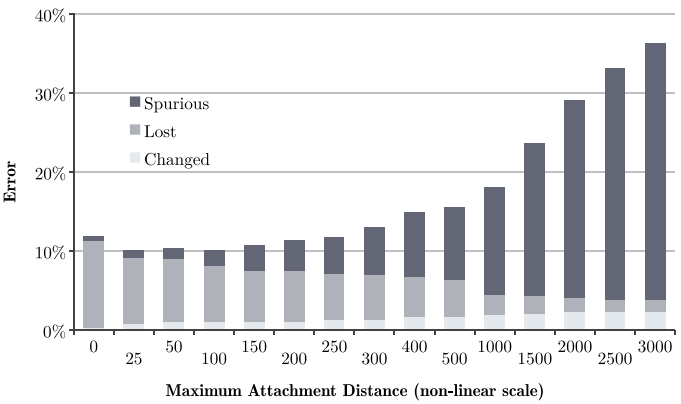


Fig. 5. Error rate of the comment attachment algorithm for different maximum attachment distances.

The results are shown in Fig. 5. Overall, the lowest error rates are to be found where the maximum attachment distance is low, which causes the number of spurious attachments to be very low. As the maximum attachment distance increases, so does the number of spurious attachments, while the number of lost attachments decreases. Interestingly, the number of changed attachments increases as well, but is very low over

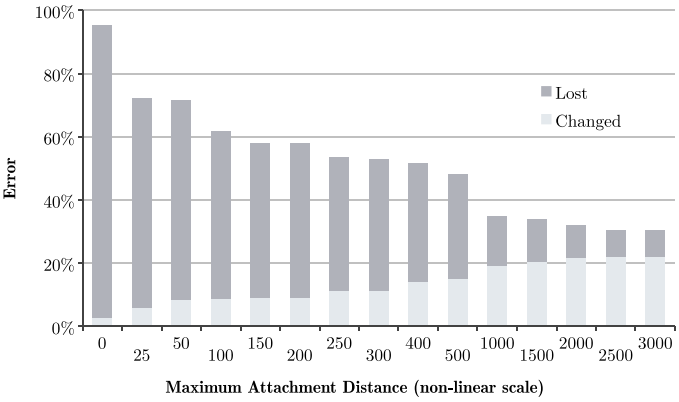


Fig. 6. Error rate of the comment attachment algorithm for different maximum attachment distances, constrained to attachments present in the reference attachment function (84 of the 308 diagrams selected for the evaluation). Note that spurious attachments are not shown here since such attachments are by definition not in the set of attachments from the reference run.

the whole range.

These results include many comments that were left unattached in the reference attachment. If we focus our attention on comments $c \in C$ with $\mathrm{att}_{ref}(c) \neq \perp$, we get the results shown in Fig. 6. Since the number of comments with attachments is lower than the number of comments without attachments, the error rate is expectedly higher. The trends remain, however: as the maximum attachment distance increases, the number of lost attachments decreases, and the number of changed attachments increases.

Interpretation

The low number of changed attachments suggests that users indeed seem to place comments closest to the vertices they relate to. How close a comment has to be to a vertex to signify an attachment, however, tends to vary: as the number of lost attachments decreases, the number of spurious attachments increases. Indeed, we have already seen that very close proximity can be found even between comments and vertices that are not at all related. The maximum attachment distance should be selected based on what should be most avoided: lost or spurious attachments.

A limitation of this experiment is that it only focusses on Ptolemy diagrams created by a limited set of developers. The extent to which the results generalize to other visual languages has yet to be determined.

V. Conclusion

We have presented a simple distance-based algorithm for computing attachments between comments and vertices in diagrams. These attachments can be used by automatic layout algorithms to place comments and attached vertices in close proximity, thus preserving visual cues for their attachment. The algorithm as implemented is currently restricted to comment-vertex attachments, but could easily be extended to include attachments to ports and edges as well.

In our evaluation, we ran the algorithm on a set of Ptolemy diagrams, with success rates of up to 90%. Thus, there is room for improvement.

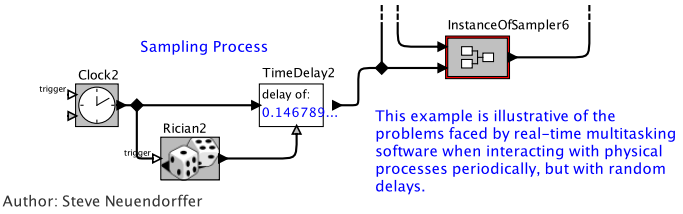


Fig. 7. Part of a Ptolemy diagram where one comment, "Sampling Process", describes a whole group of vertices while the other comment describes the diagram as a whole. It would be challenging for attachment algorithms to understand the difference since the two comments are similarly placed.

First and most importantly, our definition of proximity could be enhanced to use a weighting function that takes the alignment between comments and vertices into account. A comment placed beside a vertex should probably be attached to that vertex, even if one of the comment's corners is closer to another vertex.

Second, further attachment cues could be included. Conventions of visual languages would lend themselves well to the task since they are comparatively easy to integrate. In the case of Ptolemy diagrams, for example, comments containing the names of authors can be detected both by their position and through the fact that they contain the word *author*.

Third, more diagram elements should be considered for attachment. While the vast majority of comments in the Ptolemy example models relate to vertices, we have found a number of diagrams where comments relate to ports and edges. Comments are also often used to describe groups of vertices, as shown in Fig. 7. It is not immediately apparent how such attachments can be discovered.

Finally, if a comment mentions the name of a vertex, this might be a cue worth considering.

Acknowledgments

We thank Edward A. Lee for his kind permission to use the Ptolemy diagrams as examples. We also thank the reviewers for their valuable suggestions.

References

[1] M. Petre, "Why looking isn't always seeing: Readership skills and graphical programming," *Communications of the ACM*, vol. 38, no. 6, pp. 33–44, Jun. 1995.

[2] L. K. Klauske, "Effizientes Bearbeiten von Simulink Modellen mit Hilfe eines spezifisch angepassten Layoutalgorithmus," Ph.D. dissertation, Technische Universität Berlin, 2012.

[3] K. Misue, P. Eades, W. Lai, and K. Sugiyama, "Layout adjustment and the mental map," *Journal of Visual Languages & Computing*, vol. 6, no. 2, pp. 183–210, Jun. 1995.

[4] G. W. Klau and P. Mutzel, "Combining graph labeling and compaction," in *Proceedings of the 7th International Symposium on Graph Drawing (GD'99)*, ser. LNCS, vol. 1731. Springer, 1999, pp. 27–37.

[5] K. G. Kakoulis and I. G. Tollis, "A unified approach to labeling graphical features," in *Proceedings of the Fourteenth Annual Symposium on Computational Geometry*, ser. SCG '98. New York, NY, USA: ACM, 1998, pp. 347–356. [Online]. Available: http://doi.acm.org/10.1145/276884.276923

[6] C. D. Schulze, M. Spönemann, and R. von Hanxleden, "Drawing layered graphs with port constraints," *Journal of Visual Languages and Computing, Special Issue on on Diagram Aesthetics and Layout*, vol. 25, no. 2, pp. 89–106, 2014.