INSTITUT FÜR INFORMATIK

On Comments in Visual Languages

Christoph Daniel Schulze, Christina Plöger,
and Reinhard von Hanxleden

Bericht Nr. 1602
April 2016
ISSN 2192-6247

CHRISTIAN-ALBRECHTS-UNIVERSITÄT

ZU KIEL

Department of Computer Science
Kiel University
Olshausenstr. 40
24098 Kiel, Germany

On Comments in Visual Languages

Christoph Daniel Schulze, Christina Plöger,
and Reinhard von Hanxleden

Report No. 1602
April 2016
ISSN 2192-6247

E-mail: {cds,cpl,rvh}@informatik.uni-kiel.de

An abridged version of this work is published at the
9th International Conference on the Theory and Application of Diagrams,
Philadelphia, USA, August 2016.

Abstract

Visual languages based on node-link diagrams can be used to develop software and, like textual languages, offer the possibility to write explanatory comments. Which node a comment refers to is usually not made explicit, but is implicitly clear to readers through placement and content. While automatic layout algorithms can make working with diagrams more productive, they tend to destroy such implicit clues because they are not aware of them and thus do not preserve the relative placement of comments and the nodes they refer to. Implicit clues thus need to be inferred and made explicit to be taken into account by layout algorithms. This is what we call the *comment attachment problem*.

In this paper, we improve upon a previous paper on the subject [9], introducing further heuristics that aim to describe relations between comments and nodes. Based on an analysis of comment placement in a set of example diagrams, we develop a general comment attachment framework and evaluate the quality of its inferred attachments.

1 Introduction

Visual languages are in widespread use for developing software, either in addition to or at the expense of more traditional text-based languages. Languages such as the Unified Modeling Language (UML) complement textual languages by visualizing the architecture of a software system or the interaction of its components. In the automotive, avionics, and embedded systems industries, visual languages are largely used to actually develop the software itself, following model-based development concepts. Languages such as ASCET (ETAS Group), LabVIEW (National Instruments), SCADE (Esterel Technologies), or Simulink (MathWorks) allow developers to define software systems using *node-link diagrams* such as the one in Figure 1.1: *nodes* (or *actors*) are entities that can consume and produce data, which are transmitted between nodes through the *links* connecting them.

One reason for the popularity of visual languages may be the implicit assumption that a diagram is more easily understood than text. However, this assumption does not necessarily hold [6]. First, visual languages—much like textual languages—are for the most part general-purpose languages that offer a set of basic operators to build arbitrary systems with. While it is easy to understand what a single operator does, it is harder to figure out what a particular combination of operators is supposed to do. Note that this is true for textual languages as well: given a sufficiently complex piece of code, it can be hard to understand its purpose. Second and more fundamentally, a diagram is not easier to understand simply because it is visual. Rather, its nodes have to be carefully placed on the drawing area, and its links have to be properly routed for the diagram to be readable at all. A diagram with all of its nodes randomly scattered is practically worthless, whereas a carefully drawn diagram can be very easy to comprehend.

One way to solve the first problem in textual languages is to write comments. Which part of the code a comment describes is easy to infer: simply because of the way text works, comments are usually written above related code or at the end of a line. Many languages even offer comments for which it is explicitly defined what element they refer to, such as Java's Javadoc comments. Most visual languages also support comments, usually in the form of special kinds of nodes that display text. However, finding out which node a comment refers to can be less straightforward because of the two-dimensional nature of positioning them. Some languages solve this problem by allowing developers to explicitly attach comments to diagram elements; drawing a line between them makes their relationship very clear. But not every language supports this feature, and not every developer uses it if it does. Developers often seem to rely on other, more implicit clues instead, such as the distance between comments and nodes. This falls into the category of secondary notation [7].

The second problem mentioned above requires developers to properly position nodes

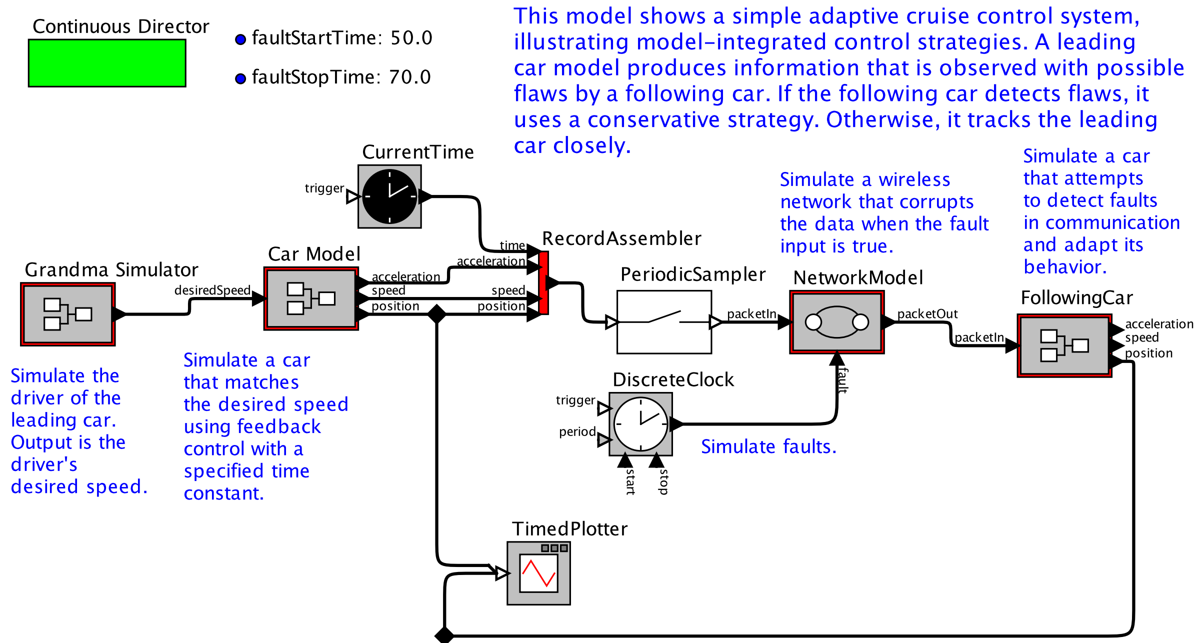

Figure 1.1: A small node-link diagram as laid out manually using the Ptolemy language. The diagram features a comment that describes the diagram in general, a comment that contains the author's name, and a number of comments that describe specific nodes. Without further processing, automatic layout algorithms will often break the implicit attachments between comments and nodes obvious to humans by placing the comments at arbitrary places.

in the diagrams they create, which is a time-consuming process. Based on observations from a study in the automotive industry, Klause and Dziobek estimated that developers spend about 30% of their time on manual layout adjustments [3]. Layout algorithms can reduce this effort by positioning nodes and routing edges automatically. However, there is a problem with automatic layout when it comes to comments: unless a comment is explicitly attached to the node it refers to, the layout algorithm has no knowledge of their relation and they may end up in vastly different places in the final layout. This wreaks havoc with the implicit clues that would have allowed a viewer to understand which node a comment refers to.

This problem can be solved in two ways. Using a layout algorithm that tends to preserve spatial relationships between diagram elements will also tend to preserve most implicit clues. Misue et al. call such layout algorithms *layout adjustment algorithms* as opposed to *layout creation algorithms*, which compute layouts from scratch [5]. However, many layout algorithms used in practice fall into the latter category. To use them in spite of their problems with comments, it is necessary to infer attachments between comments and nodes to make them explicit for the layout algorithm. This is what we introduced as the *comment attachment problem* in previous work [9].

In our experience, even if a layout algorithm generally produces good results it is problems like these that prevent users from applying it. In fact, we know of a case where the developers of a commercial diagram browsing application chose to rather hide comments than having them end up in the wrong places.

Contributions. In previous work, we evaluated how to perform comment attachment by relying only on the distance between comments and nodes [9]. The main problem there was the sheer number of comments that should have been left unattached, but were attached to nodes by the presented algorithm, which is what we call *spurious attachments*. In this paper, we introduce a number of additional heuristics that ultimately reduce the number of spurious attachments. We present an analysis of how comments are used based on these heuristics in a set of Ptolemy models. We also present a general framework for comment attachment in any language and evaluate how well it works for our use case when configured based on our analysis results. Finally, we draw conclusions on how to properly integrate comments into visual languages to support automatic layout. Note that for this paper, we limit outselves to attachments between comments and nodes and leave attachments between comments and other elements, such as edges and ports, for future work.

Use Case. Ptolemy[1] is a tool for model-based development and experimentation with actor-oriented design developed at UC Berkeley. Ptolemy supports comments in the form of nodes that contain text, and eventually added support for explicitly attaching a comment to the node it is supposed to refer to. However, most existing models do not use that feature, either because the developer does not know about it, or because the model was developed before explicit attachments were introduced. Similar to other visual languages, Ptolemy allows the behaviour of a node to be defined by a model nested inside of it. Nested models can be opened and edited in separate windows, which can make it a challenge to keep an overview of the software system.

The KIELER Ptolemy Browser[2] allows users to browse through a Ptolemy model along with its nested models in a single window by dynamically expanding or collapsing nodes. This requires automatic layout algorithms since the additional space occupied by expanded nodes requires surrounding nodes to move accordingly. The layout algorithm we use [8] is a layout creation algorithm based on the hierarchical layout method first introduced by Sugiyama et al. [10]; we therefore need comment attachment to keep comments not explicitly attached to a node close to the nodes they implicitly refer to. Note that the relevance of comment attachment is not limited to hierarchical layout algorithms, though, but applies to all layout creation algorithms.

Ptolemy ships with a set of demo models intended to showcase different models of computation, actors, and development techniques available in Ptolemy. 348 of them, created by 40 different developers, will serve as our main data set throughout this paper. All of these models are mostly based on data flow diagrams, but some contain small

[1] `http://ptolemy.eecs.berkeley.edu/ptolemyII/`
[2] `http://rtsys.informatik.uni-kiel.de/kieler`

submodels that are state machines. However, since the heuristics we present here do not require a particular layout style, they are appricable to both paradigms. Overall, the models contained 7447 nodes— resulting in an average of 21.4 nodes per model— as well as 1078 comments— 3.1 comments per model— of which 182 (about 17%) refer to a specific node. A model was included in our data set if the Ptolemy Browser was able to properly open it, it contained at least one comment, all comments refer to at most a single node, and there were no explicit attachments.

Related Work. We are not aware of any studies on how developers use comments in visual languages. The situation is different when it comes to textual languages. The usage of documentation systems such as Javadoc have indeed been studied, for example by Kramer [4], but the results do not seem to be applicable to our domain: Javadoc has clear rules on what comments refer to, which visual languages usually lack.

The heuristics we use to characterize comment usage are derived in part from our experience of looking at diagrams. Heuristics such as proximity, however, are rooted in Gestalt psychology as introduced by Wertheimer [11]. Alignment or font-size based heuristics are based on established principles in graphic design.

To the best of our knowledge, our previous paper is still the only one on the subject of inferring comment attachments in visual languages [9]. Eichelberger recognizes that comments can relate to different elements (or none at all) in UML class diagrams [2]. Other work based on textual languages, for example by Buse and Weimer on automatically augmenting Javadoc comments [1], also requires knowledge about relations between comments and code. However, the attachment rules for documentation in textual languages are usually clearly defined, not as ambiguous as in visual languages. With comment attachment, such applications may become viable for visual languages as well.

It is worth noting again that many visual languages already support explicit attachments between comments and nodes. However, not all languages do so, and not all developers make use of them.

Outline. In the next section, we will introduce our comment attachment heuristics and use them to analyse how comments are used in our set of demo models. We will then present our comment attachment framework in chapter 3 and derive a configuration for it based on the analyses. After an evaluation of the results in chapter 4, we conclude the paper in chapter 5 with suggestions for developers of visual languages and with open topics for future research.

2 Heuristics and Analysis

When looking at diagrams, it quickly becomes apparent that we can distinguish two categories of comments: *node comments* refer to a specific node while *non-node comments* do not. Non-node comments can be further divided: *title comments* contain the title of a diagram, *author comments* contain the names of a diagram's authors, and *general comments* contain general information about a diagram not specific to any one node. The goal of any *automatic attachment* is to attach every node comment to the node it refers to while leaving non-node comments unattached.

To assign each comment in our data set the proper category, we defined a *manual attachment*. That attachment was produced by manually looking at each comment and figuring out which node, if any, it refers to. If that was not clear enough, the model was removed from our data set. Usually, however, making attachment decisions did not prove much of a problem. This manual attachment is what we will be using as our reference throughout the rest of the paper.

In the following, we will introduce the basic idea of each heuristic, analyse our data set, define the heuristic based on the analysis, and evaluate how well it performs. The heuristics fall into two categories: *filters* aim to recognize non-node comments to prevent them from being attached to anything, and *regular heuristics* try to estimate how likely it is that a comment should be attached to a given node.

2.1 Font Size Filter

The aim of this filter is to recognize title comments and prevent them from being attached. Text documents usually start with a title set in a larger font size than the rest of the text. One may well hypothesize title comments in diagrams to be set in a larger font size as well, such as in Figure 2.1.

Analysis. 57 out of 348 demo models contain a title comment. Title comments always appear on the uppermost hierarchy level, and are never the only comment there, except for a single case. In 45 of the 57 cases, the title comment has the largest font size out of all comments, and its font size exceeds the default font size used for comments in Ptolemy. There is one model where a comment is the only one with the largest font size, but is not a title comment.

Heuristic (Font Size Filter). Find the set of comments on the uppermost hierarchy level with the largest font size. If the set only contains a single comment, select it as

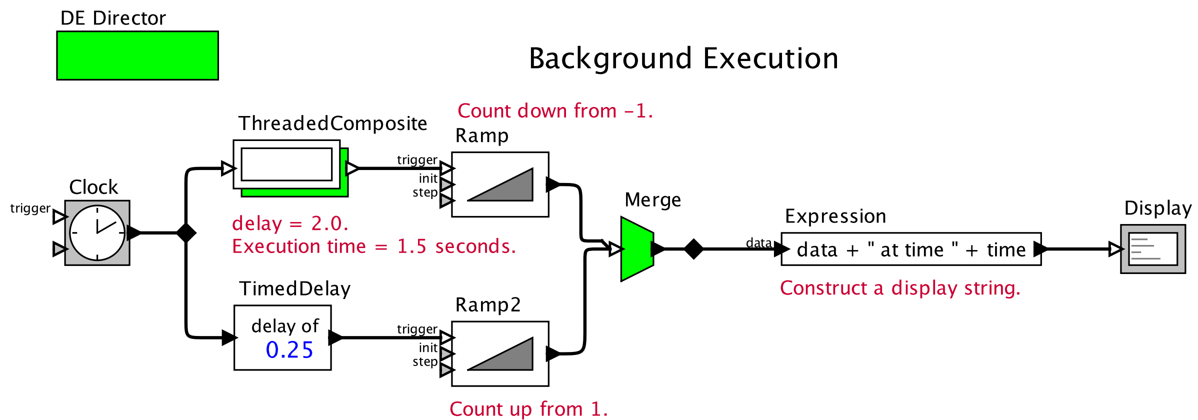

Figure 2.1: A diagram with a title comment, "Background Execution." Also note how the leftmost comment is nearer to "Ramp" than to "ThreadedComposite" in terms of their bounding boxes. The alignment, however, makes it clear that it actually refers to the latter.

the title comment and thus filter it out, provided that it is not the only comment on the uppermost hierarchy level and that its font size exceeds the default font size.

Evaluation. As expected, the font size filter finds 45 out of the 57 title comments and only once generates a *false positive* (that is, filters out a comment which actually is not a title comment). It misses the remaining title comments, thus producing 12 *false negatives*, but it seems difficult to divise a simple rule that can recognise title comments based on the semantics of the text.

2.2 Text Prefix Filter

The aim of this filter is to recognize author comments and general comments and prevent them from being attached. Often enough, programs written in textual languages contain a general description of what the program does, as well as the name of the developer who wrote the code. Some programming languages, such as Java, have built-in documentation systems to support this. Visual languages usually do not, but it seems reasonable to hypothesize that author comments and general comments will often start with similar phrases.

Analysis. The majority of models contain general comments. Many of them start with one of the following—rather sensible—prefixes: "This model," "This submodel," "This example," and "Model of." The remaining ones do not share any sensible common prefix.

Almost every model in our set contains an author comment. Except for four of them, they all start with one of three prefixes: "Author:," "Authors:," and "Demo created by."

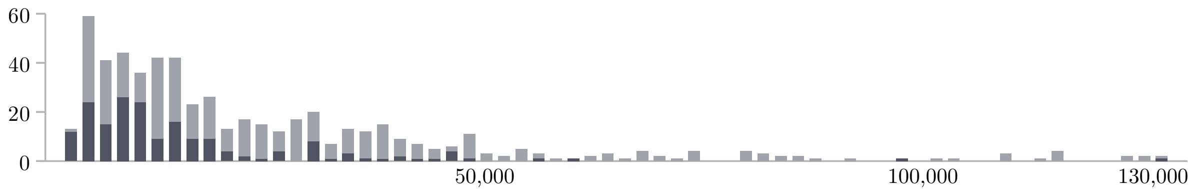

Figure 2.2: Histogram of the area in square pixels of all comments not filtered out by the font size and text prefix filters. The dark area of each bar indicates the fraction of node comments. To keep the histogram readable, it is capped at an area of 130,000 square pixels, which few comments exceed.

Heuristic (Text Prefix Filter). Filter out comments that start with one of the prefixes listed above.

Evaluation. Out of 1078 comments, the filter filters out 457 comments, producing no false positives. However, it did miss 382 author and general comments.

2.3 Area Filter

The aim of this filter is to recognize general comments and prevent them from being attached. It seems reasonable to assume that general descriptions of what a program does will often be longer—and therefore larger—than more specific comments.

Analysis. On average, node comments are indeed smaller than non-node comments (14,075 and 26,028 square pixels, respectively). However, the histogram in Figure 2.2 shows a considerable overlap between the area of node comments and comments not filtered out by the two previous filters, suggesting that our hypothesis does not work as well as expected in practice.

Heuristic (Area Filter). Filter out a comment if its area exceeds a predefined threshold.

Evaluation. As the analysis hinted at, the area filter is not very effective in filtering out general comments if the amount of false positives is to be kept low. Setting the threshold too low will generate too many false positives and will thus prevent too many comments from being attached (for instance, a threshold of $14,000$ square pixels will filter out about 68% of general comments, but will also incorrectly filter out 40% of node comments). Setting the threshold too high considerably limits the filter's effectiveness (a threshold of $46,000$ square pixels only filters out 5% of node comments, but will only filter out 20% of general comments).

Still, with a high enough threshold, at least some general comments will be filtered out without too many false positives.

	Strict Match		Fuzzy Match
	Case-Sensitive	Case-Insensitive	Case-Insensitive
Count	191	208	225
Non-node comments	124	139	155
Attached to referenced node	59	59	60
Attached to different node	8	10	10

Table 2.1: The numbers of comments out of 1078 that contain the name of a single node, along with how they are attached in the manual attachment.

2.4 Node References

The aim of this heuristic is to recognize node comments and attach them to the correct node. If the name of a node appears in a comment, we consider this to be a *node reference*. If a comment contains such a node reference, it seems sensible to assume that it should be attached to that node. This ceases to be true once further references occur in the comment: since the layout algorithm applied in our use case only allows comments to be attached to a single node, we consider such comments to be general comments.

Analysis. Table 2.1 contains the results of an analysis of node references in comments. We distinguish three kinds of matchings: strict matching requires the node name to appear in the comment as is, either observing or disregarding case. Fuzzy matching relaxes this constraint: it allows arbitrary whitespace to appear between the words a node's name consists of, including the different components of "CamelCased" node names.

Note how most comments that mention a node are not actually attached to it in the manual attachment, such as the comment in Figure 2.3. These comments fall into two categories: first, general comments often mention a single node which is of particular importance to the model; and second, comments sometimes compare "this" node or model to a node mentioned by name, which it of course should not be attached to—after all, it mainly refers to "this" node or model.

An interesting hypothesis is to assume that there is a distance threshold above which commenst that reference a node are not attached to that node in the manual attachment. To test this hypothesis, Figure 2.4 shows a histogram of the distance between a comment and the node it is referencing. As it turns out, there is no specific distance threshold above or below which all references are correct according to the manual attachment. Still, most correct references seem to accumulate in the lower distances. Indeed, in about 50% of cases, comments attached to the node they mention are also closest to that node.

Heuristic (Node Reference Heuristic). If a node name appears exactly as is in the text of a comment, attach the two unless the comment contains the names of other nodes

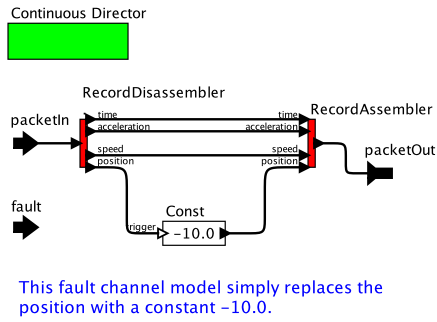

Figure 2.3: The comment in this diagram is a general comment explaining the diagram as a whole, but contains the name of the "fault" input port on the left. Such comments cause the node reference heuristic to produce false positives.

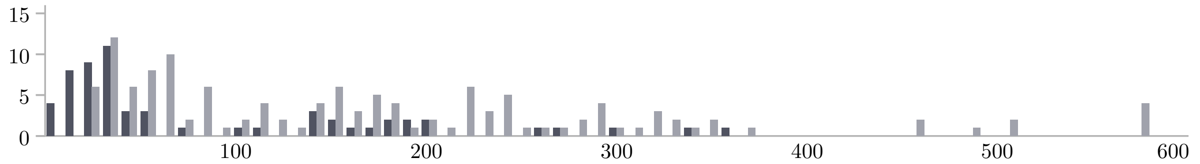

Figure 2.4: Histogram of the distance between comments and the node they mention, for cases where they are also attached to that node in the manual attachment (dark) and for cases where they are not attached to it in the manual attachment (light).

as well.

Evaluation. Strict, case-sensitive matching recognises almost all attachments while producing the lowest amount of false positives among the three variants. Since the heuristic will act on the filtered set of comments in practice, false positives involving comments that describe the whole diagram will be reduced. However, comments that make references to "this node" and mention the name of another node are not attached correctly.

2.5 Distance

The aim of this heuristic is to recognize node comments and attach them to the correct node. The distance between a comment and a node may be the most obvious heuristic and was already examined in our first paper on the subject [9]. The hypothesis here is that the node a comment refers to is the one closest to the comment.

Analysis. Table 2.2 contains the results of an analysis of the distance between comments and nodes. First, note how the average distance between node comments and the node they are attached to is larger than to the node they are closest to. Also note how the distance between filtered non-node comments and the node they are closest to

	Attached Node	Closest Node		
		Attached	Filtered All	Filtered Unattached
Count	182	115	575	393
Minimum Distance	0	0	0	0
Maximum Distance	545	151	585	585
Average Distance	78.48	27.42	57.18	63.33
Standard Deviation	109.42	28.32	65.75	69.70

Table 2.2: The distance between comments and nodes they are actually attached to in the manual attachment, as well as between various kinds of comments and the node closest to them. *Filtered all* refers to comments not filtered out by the font size and text prefix filters, while *filtered unattached* further reduces that to those comments unattached in the reference attachment.

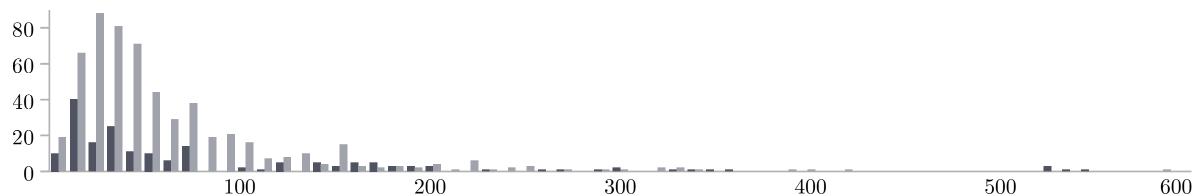

Figure 2.5: Histogram of the distance between all comments not filtered out by the font size and text prefix filters and their closest nodes (light) as well as between node comments and the nodes they are attached to according to the manual attachment (dark).

is on average smaller than the distance between node comments and the node they are attached to.

Out of 182 attached comments, 115 (63%) are actually attached to the node closest to them. Accordingly, 963 out of 1078 comments overall are not attached to the node closest to them.

Heuristic (Distance Heuristic). Find the node closest to a given comment. Attach them unless their distance exceeds a predefined threshold.

Evaluation. As the histogram of comment-node distances in Figure 2.5 suggests, it is difficult to find a good threshold value. Setting it too low will lead the heuristic to miss a lot of attachments (a threshold of 20 will miss about 73%); setting it too high will produce a lot of false positives (a threshold of 200 will only miss about 10% of attachments, but will attach about 94% of comments that should be unattached).

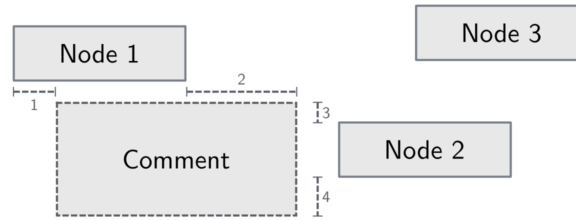

Figure 2.6: How the alignment between a comment and a node is computed differs depending on where the node is in relation to the comment. For nodes above or below the comment, such as “Node 1,” we take the minimum of the distances that keep the two from being perfectly left-aligned (1) or right-aligned (2). For nodes right or left of the comment, such as “Node 2,” we take the minimum of the distances that keep the two from being perfectly top-aligned (3) or bottom-aligned (4). Nodes that are cater-cornered to the comments, such as “Node 3,” are considered not to be aligned at all.

2.6 Alignment

The aim of this heuristic is to recognize node comments and attach them to the correct node. In graphic design, alignment between elements is used as a means to establish a relationship between them. It seems reasonable to assume that comments are aligned to the node they should be attached to, as is the case in Figure 2.1.

Analysis. The bounding box of a node can be left-, right-, top-, or bottom-aligned to the bounding box of a comment. Whatever it is, a certain distance will usually keep it from being perfectly aligned (see Figure 2.6). It is the smallest such distance over the four possible alignments that we define as the alignment value for a given comment-node pair. If a node is cater-cornered to a comment, we consider the two to not be aligned at all. Note that this has nothing to do with traditional text alignment as used in word processors.

A histogram of alignment values looks very similar to the distance histogram in Figure 2.5. 82 out of 182 node comments (45%) are attached to their best-aligned node. The average best alignment value between a node comment and any node is much better than the average alignment between the comment and its attached node (8.83 and 27.03 pixels, respectively). This makes sense, since we have not imposed a limit on their distance: it is likely that for any given comment we will find a well-aligned node somewhere in the diagram that does not necessarily have any relation to the comment.

Heuristic (Alignment Heuristic). For a given comment, find the node best aligned to it, optionally restricted to nodes within a certain maximum distance. Attach the two unless the alignment exceeds a predefined threshold.

Evaluation. As already expected from the analysis, the alignment heuristic does not fare very well. Without any restrictions on the maximum distance allowed between

comments and nodes, it will find less than 50% of correct attachments and produces a lot of false positives. Even with a distance restriction, results do not improve much.

2.7 Discussion

Based on the analyses, it seems to us that established conventions such as a big font size or how the list of authors is to be included in a diagram work best for comment attachment. Other heuristics work to an extent (node references, distance) or have little predictive value (area, alignment). A considerable share of the information that help link comments to nodes still seems to be in a comment's text. While it may for example be placed in the rough vicinity of the node it refers to, distance alone has its limits in how successfully it predicts attachments.

There are two limiting factors to this analysis. First, the data set is smaller than we would like it to be. The number of diagrams is comparatively low (348), as is the number of authors that produced the diagrams (40). Also, the number of comments actually attached in our manual attachment (182 out of 1078, 17%) is not that high. The second and more severe problem is that all diagrams were created as demonstration models for the Ptolemy tool to help explain how certain actors or models of computations are used and how to develop using Ptolemy; the heuristics that work well for this particular set of diagrams are not guaranteed to work well for another set. In fact, from looking at diagrams produced with other languages and by other developers it seems that we may not find a universally applicable set of rules for comment attachment. We feel confident, however, that our heuristics are a good starting point to analyse the usage of comments in every visual language.

3 An Attachment Framework

We have implemented the concepts presented so far in a general framework for comment attachment as part of the KIELER open source project. The framework is being used for the KIELER Ptolemy Browser, but our aim was an architecture general enough for it to be used for comment attachment in any language. Figure 3.1 shows the framework's general architecture. Customizable behaviour is encapsulated into interfaces for which the framework provides default implementations. Using the framework requires obtaining a `CommentAttacher` instance, configuring it with an appropriate set of interface implementations, and triggering comment attachment.

Most interfaces include a preprocessing method that is executed before the comment attachment algorithm itself is started. For example, the font size-based filter uses the preprocessing to find the comment with the largest font size.

The comment attachment algorithm executes the following steps for each comment:

1. The `IExplicitAttachmentProvider` is queried for a node the comment may have been explicitly attached to by the diagram's author, provided that the visual language supports explicit attachments. If there is such a node, the comment is attached to it without executing the rest of the algorithm.

   Since Ptolemy supports explicit attachments, we use a custom implementation in the Ptolemy Browser.

2. All registered `IEligibilityFilter` instances are asked whether the comment is eligible for attachment. If at least one filter determines that it is not, the comment is left unattached.

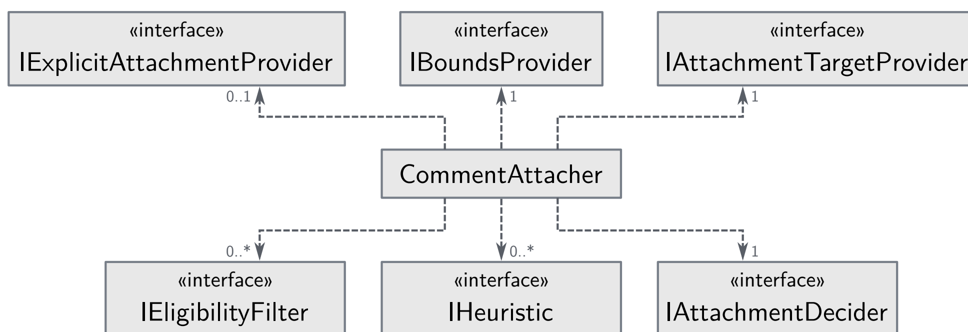

Figure 3.1: The comment attachment framework's general architecture. Its behaviour can be customized by providing implementations of the interfaces specialized to the respective use case.

The framework provides filters based on a comment's area as well as on its textual prefix. Both are used in the Ptolemy Browser, along with a custom filter that recognizes title comments both based on font size and based on special title comments available in Ptolemy.

3. The `IAttachmentTargetProvider` is queried for the list of nodes the comment could be attached to. The default implementation simply returns all nodes on the comment's hierarchy level. Custom implementations could change this by only including nodes up to a certain distance, or by allowing other comments as possible attachment targets.

   The Ptolemy Browser uses the default implementation.

4. For each node, all registered `IHeuristic` instances are queried for their assessment (between 0 and 1) as to how likely it is that the comment should be attached to the node (0 being not likely at all, 1 being very likely). The assessment of each heuristic is recorded in a map for each node. Heuristics based on spatial properties can make use of an `IBoundsProvider` implementation to retrieve the coordinates and size of comments and nodes. The framework provides default implementations for all heuristics presented in chapter 2.

   The Ptolemy Browser uses all heuristics introduced in chapter 2 except for the alignment heuristic. It uses a custom bounds provider that retrieves the positions of diagram elements as they were in the original Ptolemy tool. The size of an element can only be approximated, though, since bounds are not persisted in the model.

5. Finally, the `IAttachmentDecider` decides which node, if any, to attach the comment to. The decision is based on the heuristic values just computed. The attachment framework provides an implementation that aggregates the heuristic values for each node with a customizable aggregator function and then selects the node with the highest aggregated value, provided that it exceeds a definable threshold.

   In the Ptolemy Browser, we attach filtered comments to a node if they reference it. Otherwise, we make attachment decisions based on the distance heuristic.

4 Evaluation

To evaluate the comment attachment framework, we compared the automatic attachments computed for our data set against the manual attachment. For each comment, we check which node it is attached to in the manual attachment and in the automatic attachment. There are four cases:

1. An attachment is *correct* if a comment is attached to the same node or left unattached in both attachments.
2. An attachment is *changed* if the comment is attached to different nodes in the two attachments.
3. An attachment is *lost* if a comment is attached to a node in the manual attachment, but is not attached to any node in the automatic attachment.
4. An attachment is *spurious* if a comment is not attached to any node in the manual attachment, but is attached to a node in the automatic attachment.

4.1 Results

Distance heuristic. The error rates of performing comment attachment based only on the distance between comments and nodes are shown in Figure 4.1a. Unsurprisingly, the results are similar to what we found in previous work on the subject [9]: as the distance threshold is increased, the overall error rate increases mainly because the number of spurious attachments increases. The number of lost comments decreases as more comments are attached to nodes.

Improved heuristics. The main problem with the previous approach is the sheer number of spurious attachments. The filters introduced in chapter 2 are meant to keep comments from being attached to anything and thus should reduce these, while the node reference heuristic should help find more correct attachments. Figure 4.1b shows the results of applying the two most unproblematic filters in terms of false positives (according to the analyses in chapter 2) as well as the node reference and distance heuristics: if the node reference heuristic finds an attachment, that attachment is applied; otherwise, the distance heuristic is invoked. Indeed, this significantly reduces the number of spurious attachments as the distance threshold is increased. More importantly, however, the node reference heuristic causes less lost attachments, at the expense of more spurious attachments in the lower threshold areas. It may well be argued that this is a worthwhile

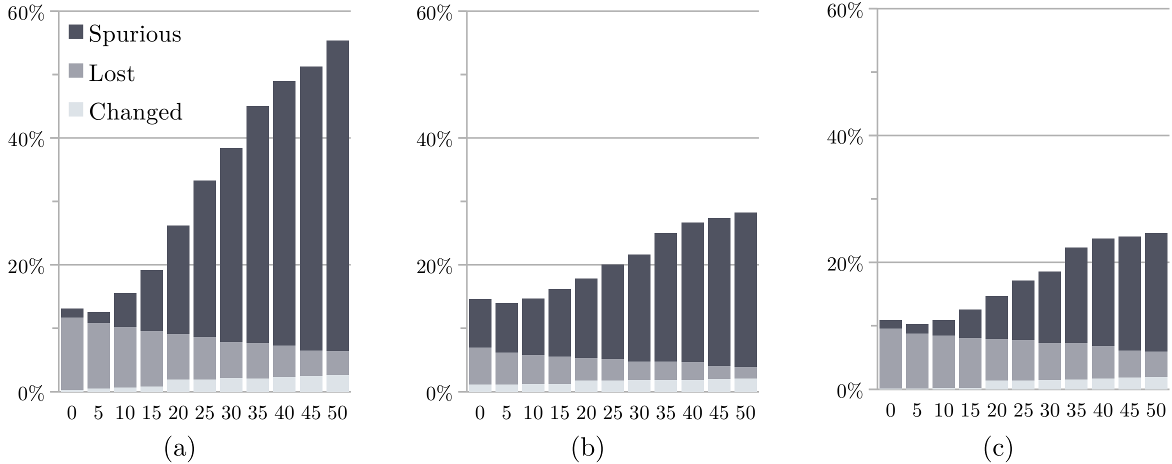

Figure 4.1: Results of comparing automatic attachments to the manual attachment, all involving the distance heuristic subject to different threshold values. (a) Automatic attachment based just on the distance heuristic [9]. (b) Automatic attachment based on the font size and text prefix filters as well as the node reference and distance heuristic. (c) Automatic attachment based on the same filters and heuristics, but with a maximum distance threshold of 30 imposed on the node reference heuristic and with the area filter with a conservative setting.

tradeoff, since a node attached to a comment by the node reference heuristic is at least mentioned in the comment.

In an attempt to further reduce the number of spurious attachments caused by the node reference heuristic, Figure 4.1c shows the results of applying a distance threshold of 30 to the node reference heuristic, and of engaging the area filter with a very conservative setting. This decreases the amount of spurious attachments and the error rate overall to about 10% at best, at the expense of found correct attachments as the number of lost attachments increases. Which result is preferrable depends on the application.

4.2 Discussion

As the evaluation shows, comment attachment can work very well, with error rates as low as about 10%. However, this requires the involved heuristics to be configured correctly: finding the maximum attachment distance that works best for a given set of models, or working out the best area threshold for a comment to be considered a general comment. There are other problems that are harder to make decisions on based on purely syntactic information, among them being whether a comment that mentions a given node actually refers to that node, to another node, or to no specific node at all. These are issues that reduce the effectiveness of comment attachment and need to be worked on.

These results may suggest that comment attachment should best be replaced by proper support for explicit attachments in visual languages. While we indeed think this to be

the case, comment attachment stays relevant for browsing scenarios similar to our use case, where the underlying language either does not provide explicit attachments or users do not make use of them.

It is the latter problem that we think is most relevant to properly integrating comments into visual languages. Users tend to avoid using features that they find too cumbersome to use. As far as explicitly attaching comments to diagram elements goes, this can prevent tool developers from making more advanced features available that are based on what comments refer to, such as automatic layout, semantic reasoning, or even generating documentation. It seems that the best solution may be twofold. First, provide different kinds of comments specialized to different content. General comments would contain general diagram information, author comments (which could be automatically inserted by development tools when creating new diagrams) would contain information about who developed the model, and node comments would contain additional information about a node. Dragging a node comment onto the drawing area could then include displaying "attachment lines" that indicate which node the tool will interpret the comment to refer to, thus forcing explicit attachments.

The addition of such features offer another area of application for comment attachment. Opening diagrams that do not make use of explicit attachments would trigger comment attachment and present the user with an automatically inferred attachment that they can then modify.

5 Conclusion

Building on our experience from previous work on the subject, we analysed how developers use comments in a data set of diagrams from the Ptolemy tool. Based on our findings, we introduced a flexible framework for comment attachment and evaluated different configurations. Use cases for the attachment framework include automatic layout as well as semantic reasoning about programs written in visual languages, as has already been explored in the context of textual languages.

There are still aspects left open for future research. First, it seems necessary to analyse the usage of comments in more visual languages and compare the results. It also seems worthwhile to survey users of visual languages as to whether they use any deliberate conventions when writing and placing comments. Second, the attachment framework as well as the heuristics will have to be extended to support comments attached to diagram elements other than nodes. And third, comments sometimes describe a whole group of nodes. It seems extremely hard to infer such group attachments, but this intuition is in need of proper confirmation.

Bibliography

[1] R. P. Buse and W. R. Weimer. Automatic documentation inference for exceptions. In *Proceedings of the 2008 International Symposium on Software Testing and Analysis*, ISSTA '08, pages 273–282. ACM, 2008.

[2] H. Eichelberger. *Aesthetics and Automatic Layout of UML Class Diagrams*. PhD thesis, Bayerische Julius-Maximilians-Universität Würzburg, 2005.

[3] L. K. Klauske and C. Dziobek. Improving modeling usability: Automated layout generation for Simulink. In *Proceedings of the MathWorks Automotive Conference (MAC'10)*, 2010.

[4] D. Kramer. API documentation from source code comments: A case study of Javadoc. In *Proceedings of the 17th Annual International Conference on Computer Documentation*, SIGDOC '99, pages 147–153. ACM, 1999.

[5] K. Misue, P. Eades, W. Lai, and K. Sugiyama. Layout adjustment and the mental map. *Journal of Visual Languages & Computing*, 6(2):183–210, June 1995.

[6] M. Petre. Why looking isn't always seeing: Readership skills and graphical programming. *Communications of the ACM*, 38(6):33–44, June 1995.

[7] M. Petre. Cognitive dimensions beyond the notation. *Journal of Visual Languages & Computing*, 17(4):292 – 301, 2006.

[8] C. D. Schulze, M. Spönemann, and R. von Hanxleden. Drawing layered graphs with port constraints. *Journal of Visual Languages and Computing, Special Issue on Diagram Aesthetics and Layout*, 25(2):89–106, 2014.

[9] C. D. Schulze and R. von Hanxleden. Automatic layout in the face of unattached comments. In *Proceedings of the IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC'14)*, Melbourne, Australia, July 2014.

[10] K. Sugiyama, S. Tagawa, and M. Toda. Methods for visual understanding of hierarchical system structures. *IEEE Transactions on Systems, Man and Cybernetics*, 11(2):109–125, Feb. 1981.

[11] M. Wertheimer. Untersuchungen zur Lehre von der Gestalt. II. *Psychologische Forschung*, 4(1):301–350, 1923.