Improving Comment Attachment Algorithms

Christina Plöger

Bachelor Thesis
2015

Real-Time and Embedded Systems Group
Prof. Dr. Reinhard von Hanxleden
Department of Computer Science
Kiel University

Advised by
Dipl.-Inf. Christoph Daniel Schulze

Eidesstattliche Erklärung

Hiermit erkläre ich an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Kiel,

Abstract

Visual languages are regularly used in different industries. Ptolemy is an example of a visual language and allows the developer to construct complex, heterogeneous models.

During the development of a diagram, its elements often have to be rearranged after structural changes to improve the diagram's readability. To avoid this time consuming process, automatic layout algorithms are used.

Generally, comments can facilitate the understanding of diagrams. Therefore, it is desirable that automatic layout algorithms can attach comments properly.

There exists one algorithm that is able to attach comments automatically in a set of demo diagrams from the Ptolemy tool with the help of invisible cues. This algorithm bases the comment attachment on one criterion, the proximity, and yields good success rates. However, there is room for improvement.

This thesis aims at improving the success rates of automatic comment attachment in the set of Ptolemy demo diagrams by extending the already existing algorithm with: the alignment of comments, the exclusion of title comments, of comments that identify the author as well as of comments that exceed a certain size and the mentioning of labels of elements in comments. Additionally, different combinations of heuristics are presented.

In the evaluation, the success rates of automatic comment attachment of two combinations of heuristics are examined. The first combination consists of the alignment heuristic and the heuristics that exclude comments from the attachment process. The second combination also takes into account the heuristic that deals with the mentioning of labels in comments. These combinations are compared with the already existing proximity heuristic and the case where no comment attachment is applied. Both combinations achieve very similar results and improve the success rates of the proximity heuristic. However, the combinations of heuristics only yield slightly better results than the heuristic where no comment attachment is applied.

Contents

Chapter 1

Introduction

Visual languages are regularly used in different industries such as the automotive or aerospace industry. Ptolemy,[1] as one of those languages, offers developers the possibility to model and design complex, heterogeneous systems [EJL+03].

The models produced with the help of visual languages are often based on node-link diagrams. Nodes (or vertices) process data through ports, whereas links (or edges) transfer these data between the ports. Figure 1.1 shows a diagram from the Ptolemy framework.

In general, diagrams are valued for their good readability. They are often assumed to be easier to understand than textual representations [SvH14]. However, the readability of a diagram highly depends on its layout [KD10].

In visual as well as in textual programming languages, comments can facilitate the understanding of a diagram or source code. First, they help programmers to understand diagrams or code developed by someone else. Second, they serve as a memory aid for the programmer when he takes a second look at his work after some time [AG98].

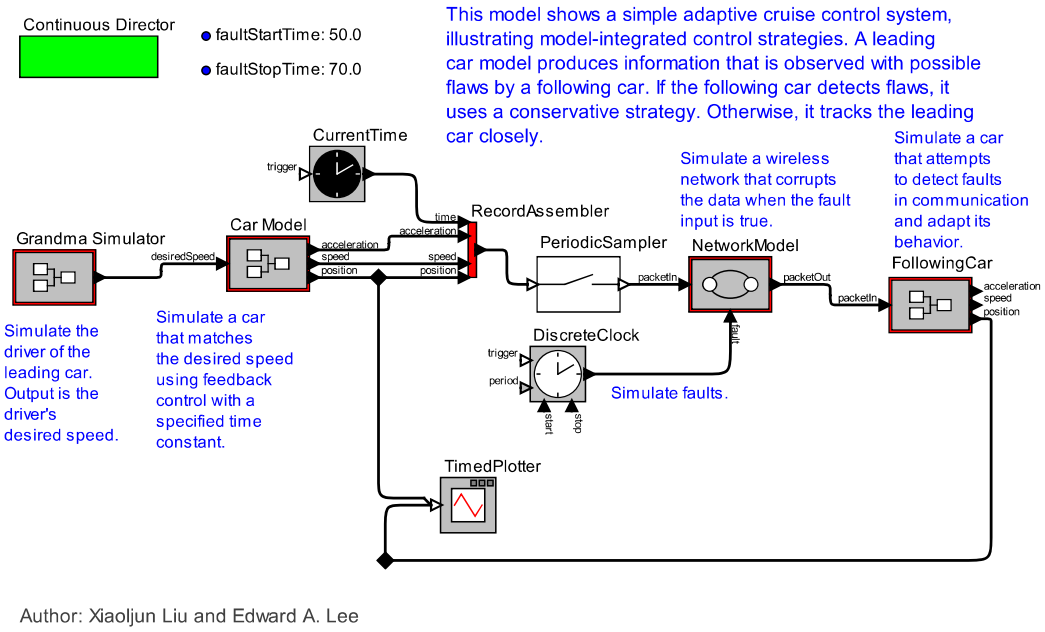
In Figure 1.1, "Author: Xiaoljun and Edward A. Lee" and "Simulate faults." are examples for comments. At first sight, comments seem to be very different from the other elements of the diagram because of their appearance. However, if we regard the diagram on a more abstract level, comments represent vertices.

Figure 1.1 also depicts that there are different kinds of comments. On the one hand, comments can concern the entire diagram. "Author: Xiaoljun and Edward A. Lee" is an example for such a comment. On the other hand, comments can refer to a single element. For example, "Simulate faults." belongs to *DiscreteClock*.

During the development of a diagram, its elements, such as vertices, ports, or edges, often have to be rearranged to improve the diagram's readability after a structural change [KD10]. To avoid this time-consuming process, automatic layout algorithms are used.

Misue et al. [MELS95] differentiate two types of layout algorithms, layout adjustment and layout creation algorithms. Layout adjustment algorithms only adapt the layout of the original diagram by changing some elements. Therefore, they are useful for interactive systems where small changes are regularly made. On the contrary, layout creation algorithms build the layout from scratch every time the diagram is modified. Most automatic layout algorithms are layout creation algorithms, including the algorithms that are addressed in

[1] http://ptolemy.eecs.berkeley.edu/



Author: Xiaoljun Liu and Edward A. Lee

Figure 1.1. Demo diagram that ships with the Ptolemy framework. *Grandma Simulator*, *Car Model*, *CurrentTime*, *RecordAssembler*, *PeriodicSampler*, *DiscreteClock*, *TimedPlotter*, *NetworkModel*, *FollowingCar* as well as *Continuous Director* represent vertices. Ports are illustrated by the small triangles at the sides of the entities. Links are represented by the arrows that connect the vertices.

this thesis.

To be able to use the information provided by comments, it is important that automatic layout algorithms attach comments to the elements they refer to. Figure 1.2 shows the diagram of Figure 1.1 after an automatic layout algorithm that does not consider comments was applied. As a result, it is not possible to distinguish which comment belongs to which element without a deep understanding of the diagram. Thus, the information provided by the comments is lost.

However, to attach comments to the elements they belong to, layout algorithms require some kind of connection between them. This connection must either be explicit or inferred in some way. Thus, on the one hand, the connection can be represented by a line or some other kind of visible connection, on the other hand, it can be inferred by invisible cues such as the proximity between the element and the comment [SvH14].

If comments are attached explicitly, several layout algorithms can place them near the elements they belong to. However, even though there are diagram editors that allow developers to set explicit attachments manually, this feature often is not used.

If there is no explicit attachment, it mostly is more difficult to infer the relations between

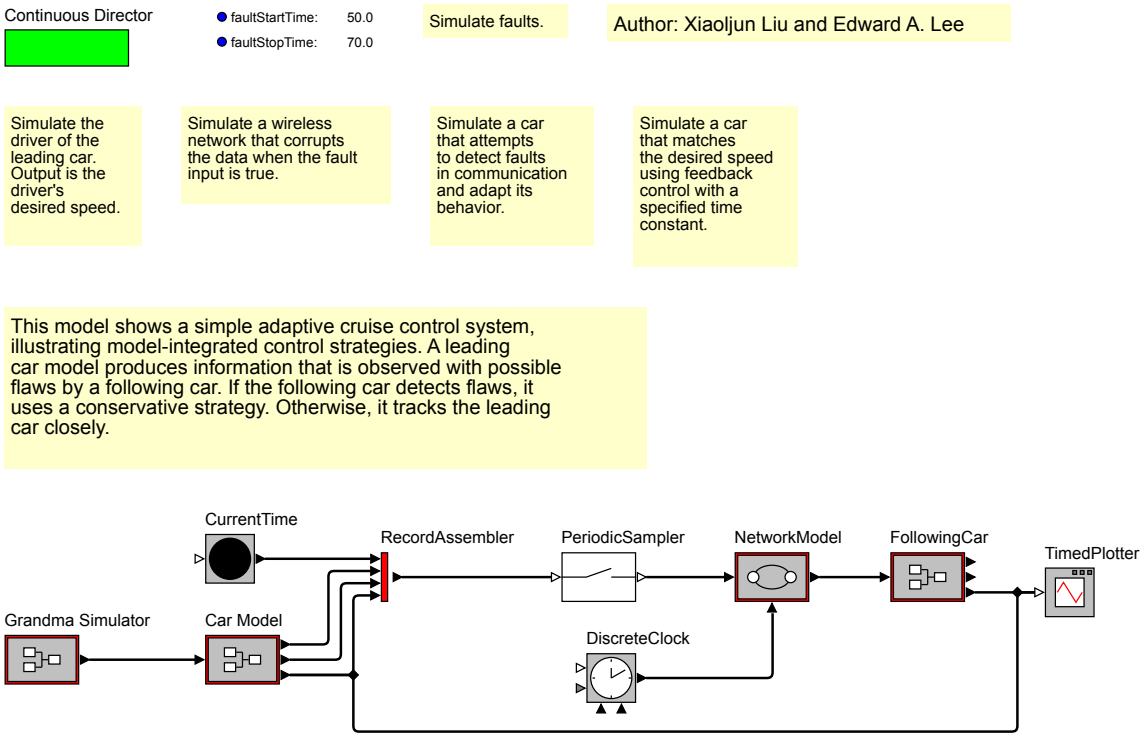


Figure 1.2. This diagram is the result of the application of an automatic layout algorithm that does not consider comments. All comments are placed at the top of the diagram.

comments and elements in diagrams than in source code. In source code, comments are usually written right before or after the line of code they relate to. In contrast, implicit attachments in visual languages are often more ambiguous [SvH14]. An example for ambiguous attachment is provided by Figure 1.3.

Without explicit attachment, many layout algorithms place comments at arbitrary positions in the diagram resulting in the loss of the information provided by the comments. Therefore, when using automatic layout algorithms, it is necessary for the proper placement of comments to find a way to detect the relations between comments and elements in the original diagram [SvH14].

1.1 Problem Statement

The goal of this thesis is to improve the success rate of automatically attaching comments to nodes by extending an already existing algorithm for comment attachment.

The existing algorithm bases the attachment on a single criterion, the proximity. That means that a comment is attached to the vertex it is closest to. During the evaluation of

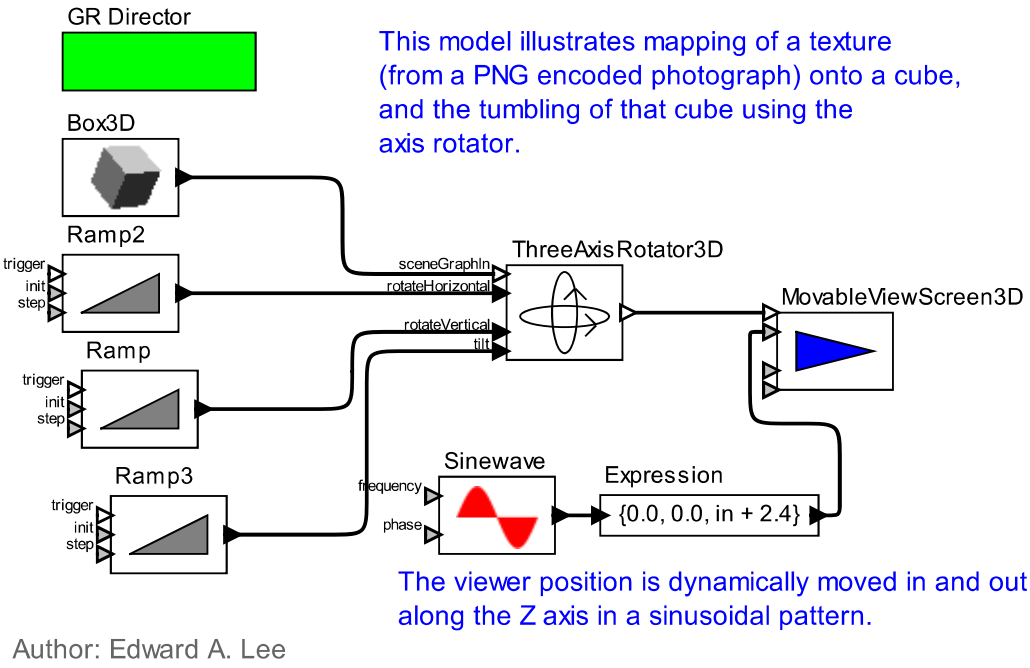


Figure 1.3. Another demo diagram from the Ptolemy framework. This diagram shows that implicit attachment of comments can be ambiguous. For the comment that is located in the bottom right of the diagram, it is not clear whether this comment refers only to *Sinewave* or to *Expression* as well.

the algorithm, success rates of up to 90% were found. However, the results were highly sensitive to configuration. Hence, there is still room for improvement [SvH14].

In this thesis, we present additional heuristics and their implementations. The heuristics discussed are alignment of comments, exclusion of title comments and comments that specify the author, mentioning of labels of elements in comments and a comment's size. We evaluate the applicability of those heuristics with a set of demo diagrams from Ptolemy and use the results to develop a combining function. Finally, the combining function is evaluated with another set of demo diagrams from Ptolemy.

1.2 Related Work

The work by Schulze and von Hanxleden [SvH14] is the basis for this thesis. They presented the algorithm mentioned above that uses proximity to attach comments. To evaluate their algorithm, they used a set of demo diagrams that ship with the Ptolemy tool.

First, they developed a reference function by manually assigning all comments to be considered to their vertices. Then, they applied their algorithm to the set of diagrams

and compared the manual to the automatic attachment. An attachment is considered to be correct or successful if the automatic attachment provides the same result as the manual attachment. As a result, they achieved success rates of up to 90%, but there are still spurious or lost attachments. Spurious attachments occur when a comment is attached by the algorithm but not by the reference function. On the other hand, the attachment is lost when a comment is attached by the reference function but not by the algorithm. The number of lost and spurious attachments depends on the *maximum attachment distance*.

The maximum attachment distance is a certain threshold value. A comment will only be attached to a vertex if the distance between the two does not exceed this value. Additionally, in order to be attached, the comment has to be the closest comment to the vertex out of all comments within the maximum attachment distance.

A low maximum attachment distance yields the best success rates with a few lost attachments and a very low number of spurious attachments. When increasing the maximum attachment distance, the success rates drop. In detail, the number of spurious attachments increases significantly, while the number of lost attachments decreases only slightly. In addition, changed attachments occur. Changed attachment means that a comment is attached to a different vertex by the heuristic algorithm than by the reference function.

To the best of our knowledge, no further work on integrating the ability to find implicit attachment of comments into automatic layout algorithms exists. However, Eichelberger discusses aesthetics of UML diagrams and presents an automatic layout algorithm for UML diagrams that includes the attachment of comments [Eic05].

His work is different in the sense that the comments in his thesis are always explicitly attached if they do not concern the entire diagram. During his discussion of aesthetics, Eichelberger states that there are different kinds of comments: comments that concern the entire diagram and comments that relate to one or more vertices. In addition, it is explained where comments that relate to one element or a group of elements should be placed [Eic05].

To be processed by the automatic layout algorithm, most comments are put into composite nodes together with the elements they relate to. Comments that concern the entire diagram are placed in the corners of the diagram. The remaining comments that are neither put into composite nodes nor concern the entire diagram are removed from the diagram [Eic05].

An important part of this thesis consists of developing heuristics for comment attachment. Regarding this topic, one of the underlying questions is: what are the cues for humans to group elements in diagrams? One of the first people to be engaged with this topic was Max Wertheimer, one of the founders of *Gestalt psychology*. In his work from 1923 he presents principles, or, as he himself calls them, *factors*, of grouping. These principles include proximity, similarity of color or size, common fate, closure, good continuation, objective set, and past experiences [Wer23].

The principle of proximity states that elements that are close together seem to belong

together. Similarity of color or size means that elements that look alike tend to be grouped. Elements are seen as a unit when they move in the same manner in the same direction. This is the principle of common fate. Closure stands for the rule that self-contained objects tend to be perceived as a unit even though they are in close contact to other objects. The principle of good continuation states that humans prolong lines in a way that fits best the form the line had before. Humans tend to persist on an initial grouping of elements, even though the elements have been moved and already imply another grouping. This principle is called objective set. Finally, humans resort to past experiences when trying to make sense of an object [Wer23].

Over the years, further principles have been proposed. One of those principles is common region, which was presented by Palmer in 1992. It states that elements that are located in one framed area are considered to be grouped [Pal92]. Uniform connectedness presents a further principle, also introduced by Palmer. It means that elements that are connected tend to be grouped together [PR94].

Of course, there also exists a great amount of work dealing with design and its principles. For example, Lidwell et al. consider alignment not just as a psychological principle, but also as a design principle [LHB03]. Elements are usually aligned in rows or columns. They state that two aligned elements imply that they are related [LHB03].

When regarding text, they argue that text blocks which are left- or right-aligned create an invisible line on the left or right. That line presents a very strong alignment cue for other elements of the design. In their opinion it is much stronger than for example cues provided by center-aligned text blocks [LHB03].

To provide another example, Williams states that elements in close proximity are perceived as a unit. Additionally, he proposes that all elements on a page should be aligned in order to yield a structured design. Alignment helps to preserve the integrity of the design even though elements are far away from each other and our mind has separated them due to other cues [Wil08].

1.3 Structure of this Document

Chapter 2 defines terminology and introduces the technologies used for this thesis.

In Chapter 3, the heuristics with which the algorithm is extended are presented. First, the role of author and title comments in diagrams is explained and the algorithms for finding such comments are discussed. Second, we define alignment of elements, present problems that come up when defining or implementing alignment and discuss the resulting algorithm. Third, several possibilities to find label names of vertices in comments are introduced. We start with an intuitive algorithm and develop two improved alternatives. Fourth, we put all the heuristics together into a function that weighs every heuristic depending on its importance for comment attachment.

Chapter 4 presents an evaluation and interprets its results. For the evaluation, the number of correctly attached comments and the performance of the algorithms are taken into account.

Finally, Chapter 5 summarizes and proposes the results of this thesis. It proposes further work that could be relevant for this topic.

Chapter 2

Preliminaries

This chapter presents the basics required to understand this thesis. Initially, important terms are defined. Afterwards, the technologies used for this thesis are explained. These include Ptolemy, the Modeling Markup Language, Xtend and KIELER. When introducing KIELER, the process of loading and displaying Ptolemy models is illustrated.

2.1 Terminology

In this section, important terms are defined to establish a basis for the upcoming chapters. Some of these definitions are based on the work of Schulze and von Hanxleden [SvH14].

Definition 1. A *graph* is a pair $G = (V, E)$ of a finite set of vertices V and a finite set of edges E, either directed or undirected.

Definition 2. A *graph with comments* is a tuple $G = (V, E, C)$ of a finite set of vertices V, a finite set of edges E, either directed or undirected, and a finite set of comments C.

Definition 3. The *x-position in the plane* of a vertex $v \in V$ or a comment $c \in C$ is a function $pos_x \colon V \cup C \to \mathbb{R}$. $Pos_x(v)$ represents the x-value of the upper left corner of v and $pos_x(c)$ the x-value of the upper left corner of c.

The *y-position in the plane* of a vertex $v \in V$ or a comment $c \in C$ is a function $pos_y \colon V \cup C \to \mathbb{R}$. $Pos_y(v)$ represents the y-value of the upper left corner of v and $pos_y(c)$ the y-value of the upper left corner of c.

Definition 4. The *width* of a vertex $v \in V$ or a comment $c \in C$ is a function $width \colon V \cup C \to \mathbb{R}$. $Width(v)$ or $Width(c)$ represents the largest horizontal expansion of v or c.

The *height* of a vertex $v \in V$ or a comment $c \in C$ is a function $height \colon V \cup C \to \mathbb{R}$. $Height(v)$ or $height(c)$ represents the largest vertical expansion of v or c.

Definition 5. A *comment attachment function* $att : C \to V$ is a possibly partial function that takes a comment and assigns the vertex it relates to, if any. If a comment $c \in C$ is not attached to a vertex, $att(c)$ is set to $att(c) = \perp$.

Definition 6. A *comment attachment algorithm* takes a graph with comments $G = (V, E, C)$ together with the width, height, and positions of each vertex and comment and computes a comment attachment function as defined above.

2.2 Used Technologies

2.2.1 Ptolemy II

Ptolemy II is an open source application developed as part of the Ptolemy project[1] by the Department of Electrical Engineering and Computer Sciences of the University of California at Berkeley [LXL01]. In the Ptolemy project, the modeling, simulation, and design of concurrent, often complex systems is researched. Particularly, embedded, heterogeneous systems are of great interest to the project [Lee03].

Consistent with the project, Ptolemy II enables the developer to design hierarchical, heterogeneous systems. The representation of hierarchical systems is possible because models can be nested inside of other models. Moreover, Ptolemy offers different models of computation [LXL01]. A model of computation determines the rules that apply to the computation of and interaction between the model's elements. An example for a model of computation is *Discrete-Events* in which elements communicate when special events occur [Lee03]. A model that contains the Discrete-Events model of computation is displayed in Figure 2.1. The model of computation is represented by the rectangle with the green background and the black border in the upper left corner. To achieve heterogeneity in systems, the sub-models of a hierarchical model can have different models of computation [LXL01].

In the framework, models are given as hierarchical graphs that consist of *entities* and *relations*. Entities are either atomic or composite entities. Composite entities contain other entities. On the contrary, atomic entities present the end of the hierarchy [LXL01]. In Figure 2.1, *ModalModel* is an example for a composite entity, while *TimedPlotter* represents a component entity. Entities have *ports*, displayed in Figure 2.1 by the triangles at the sides of *ModalModel*, for example. Relations connect entities through the ports [LXL01]. In Figure 2.1 relations are represented by the connecting lines. They usually are undirected, so information can be transported in both directions. Figure 2.1 shows another type of element on the bottom right of the diagram. These elements represented by a blue dot, a name, and a value are called *parameters*.

In this thesis, demo diagrams that ship with the Ptolemy framework are used to test the different heuristics and to evaluate the resulting combining function.

Ptolemy uses the *Modeling Markup Language* as the file format to store its models [Lee03]. Since the Modeling Markup Language causes issues during the synthesis of the diagram in KIELER, it is introduced in more detail in the next subsection.

[1] http://ptolemy.eecs.berkeley.edu/

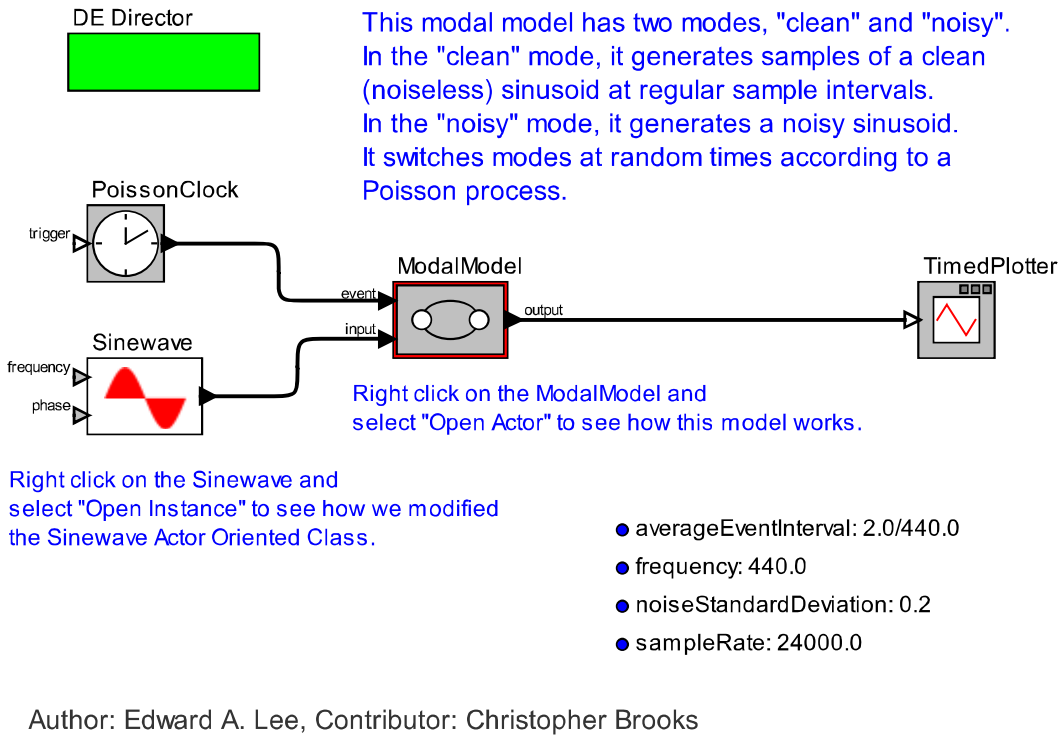


Figure 2.1. Demo diagram shipped with the Ptolemy framework. The model of computation can be seen in the upper left corner.

2.2.2 Modeling Markup Language

The Modeling Markup Language, or short MoML, is an XML-based file format [LN00]. The language is not dependent on a specific tool, but can be used by several tools if loading of the MoML files is possible. However, one prominent example of a framework that uses MoML is Ptolemy.

Every model has a top-level element that serves as the root of the model and needs to have a `name` and a `class` attribute which defines the element's type.

Typical elements of MoML models are entities which usually contain name and class attributes. The class attributes contain predefined Java classes that implement the entities. Entities can be inserted into other entities to achieve a hierarchical structure and can contain additional information. The additional information range from an element's location to the text a comment displays. The information is usually stored in a property. A property has to have a name, but can also contain all the detail information mentioned above. There are two ways how the additional information can be stored in the property.

1. The information can be saved in the `class` attribute and `value` attribute of the property. This is shown in the following exemplary code piece that is taken from the Ptolemy demo diagram *CarTracking*.

```
<entity name="Subtract" class="ptolemy.actor.lib.AddSubtract">
        <property name="_location"
                class="ptolemy.kernel.util.Location" value="[95.0, 90.0]">
        </property>
</entity>
```

2. Additional information can be included by using the *configure* element. This element offers two ways to add the information. First, it can reference an external file that does not have to be written in MoML or XML. Second, the *configure* element can contain the information directly as text. This may include markup. One example of a markup that is used in a *configure* element is *svg markup*. The exemplary code piece shows a property that contains a *configure* element and *svg markup*. It is taken from the Ptolemy demo diagram *CarTracking*. The *svg markup* is worth mentioning because the information it contains is not easy to access during the diagram synthesis in KIELER. This is explained in detail in Section 2.2.4.

```
<property name="_smallIconDescription"
class="ptolemy.kernel.util.SingletonConfigurableAttribute">
 <configure>
  <svg>
        <text x="20" style="font-size:14; font-family:SansSerif;
        fill:blue" y="20">-A-</text>
  </svg>
 </configure>
</property>
```

An entity has the possibility to specify ports which can be marked as an input, an output, or both. The marking information is stored in a property. Ports can also contain other properties to provide further information.

Additionally, entities are connected through their ports by relations. In this context, the term connection has to be distinguished from the term link. A connection associates ports, whereas a link associates ports and relations. Therefore, a connection is composed of a relation as well as at least two links [LN00]. Figure 2.2 illustrates the differences between relation, link and connection.

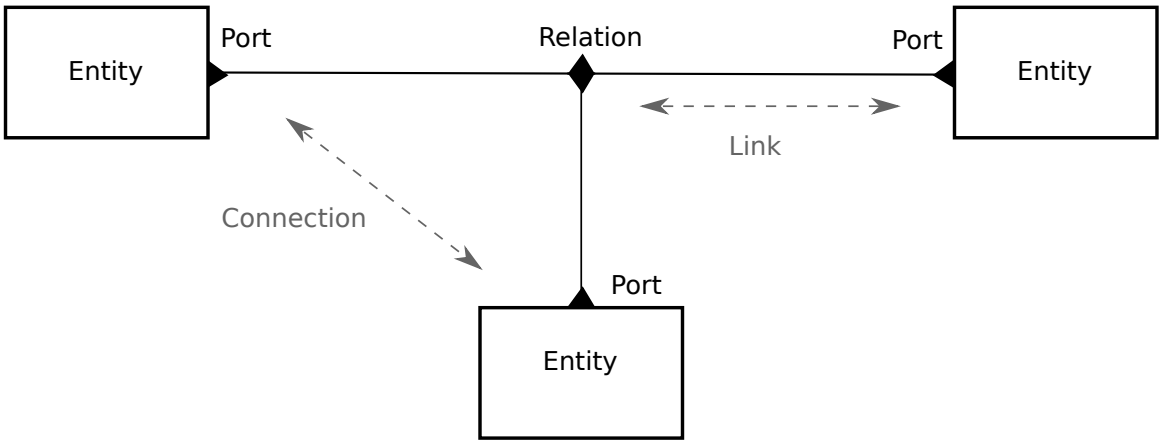


Figure 2.2. Illustration of the terminology, see [LN00].

2.2.3 Xtend

Xtend[2] is a programming language which is developed as part of the Eclipse project. It derives from the Java programming language and compiles to readable Java source code. Every Java library can thus also be used in Xtend.

Xtend eases the development of model-to-model-transformations [KPP08]. Thus, regarding the framework this thesis is concerned with, the transformation of Ptolemy models to displayable graphics is implemented in Xtend. Additionally, the process of attaching comments uses Xtend as its programming language.

2.2.4 KIELER

KIELER[3] is an open source software framework that is developed by the Real-Time and Embedded Systems Group at Kiel University. The acronym stands for Kiel Integrated Environment for Layout Eclipse Rich Client. The research project aims at improving how developers work with complex models. A major aspect of the project is the development, improvement, and application of automatic layout algorithms.

KIELER can be divided into three main parts: *Layout*, *Pragmatics* and *Semantics*. The Layout part is concerned with building layout algorithms. A central task of the Semantics part is the compilation and simulation of models. The Pragmatics part deals with the enhancement of the practical handling of graphical models. This task comprises, amongst other things, the editing or browsing of models [SFvH09].

A project from KIELER worth mentioning is KIELER Infrastructure for Meta Layout, or short *KIML*. Meta layout works under the assumption that different kinds of diagrams

[2] http://www.eclipse.org/xtend/
[3] http://rtsys.informatik.uni-kiel.de/rtsys/kieler/

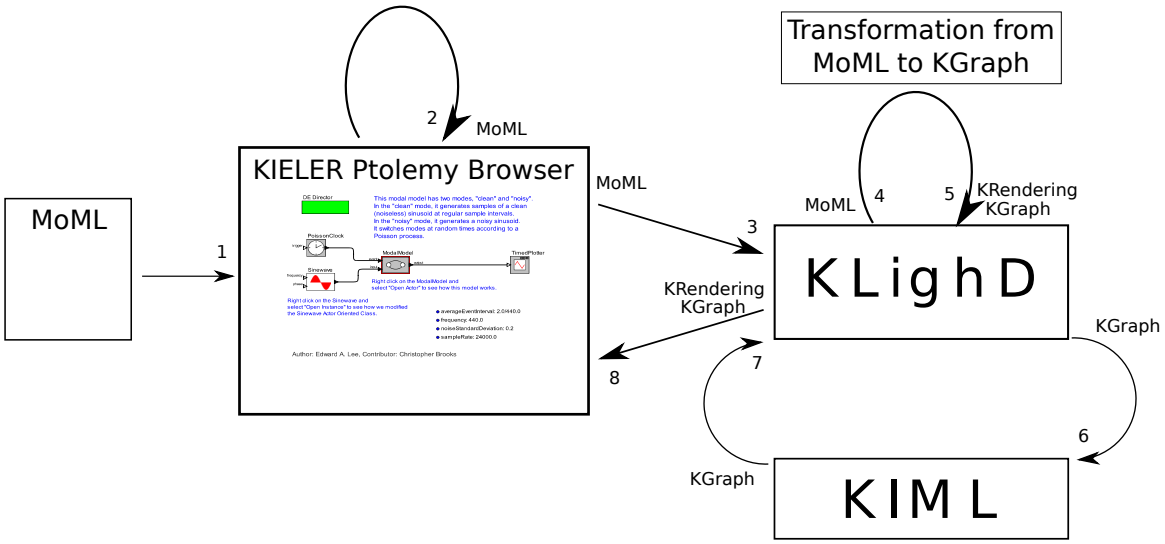


Figure 2.3. Overview of the data flow from model file to diagram, see [FvH10]. In this diagram, a model that has MoML file format is chosen as an exemplary model file. In addition, the browser for displaying Ptolemy models, called KIELER Ptolemy browser, is used as the editor. Stages of the data flow: 1) opening the editor in which the diagram should be displayed, 2) loading of an instance of the meta model that corresponds to the type of the model to be displayed, 3) sending the data to KLighD, 4) handing over the data to the correct translation by KLighD, 5) translating the data into KGraph elements and adding rendering information for every element, 6) passing the instance of the KGraph to KIML, 7) choosing a proper layout, 8) sending all the information necessary for proper display to the editor.

need different kinds of layout algorithms to be displayed optimally. Therefore, a diagram should be matched automatically with the best fitting layout algorithm [FvH10].

One important project of the Pragmatics group is KIELER Lightweight Diagrams (KLighD). KLighD aims at automatically generating and displaying diagrams from different kinds of models. To achieve this, KLighD coordinates the different parts of KIELER that are needed to display a diagram from an arbitrary model file [SSvH13]. Figure 2.3 provides an overview of the way the data travel from a model file to a displayed diagram. To be able to understand Figure 2.3, it has to be mentioned that KLighD can only display models that use a special data structure, called *KGraph*. Thus, every model has to be transformed into a KGraph before being displayed.

One element of Figure 2.3 is the KIELER Ptolemy Browser. This browser is now introduced properly.

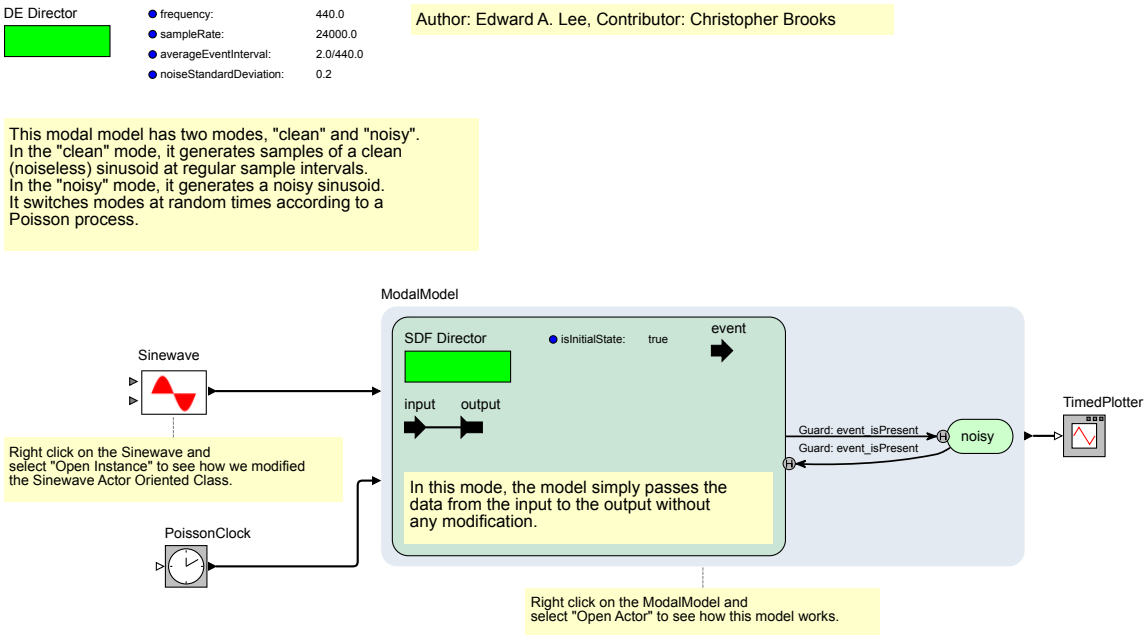


Figure 2.4. Demo diagram displayed with the KIELER Ptolemy Browser. The refinement of the *ModalModel* can be examined in-place.

KIELER Ptolemy Browser

The KIELER Ptolemy Browser[4] is an example project for the use of KLighD to visualize models in a way that they are easy to peruse. It presents Ptolemy diagrams similarly to their usual representation in Ptolemy II. However, the KIELER Ptolemy Browser differs in several points from other representations. First, it allows the developer to look inside a composite entity in-place, whereas usually a new window has to be opened. This can facilitate the understanding of the composite entity's use and how the composite entity works. Figure 2.4 shows the diagram displayed in Figure 2.1 with the *ModalModel* entity opened.

Second, there is an option to show the diagram without comments in order to focus on the elements. Third, when using the KIELER Ptolemy Browser, the algorithm for comment attachment as presented in Section 1.2 usually is applied, but can also be switched off. Fourth, another option allows different representations of edges. Finally, opened composite entities have darker backgrounds than the diagrams of their parents to stress the hierarchical structure.

As a next step, the transformation from a Ptolemy model to a KGraph is explained. To be able to understand the transformation, KGraph is introduced first.

[4] http://rtsys.informatik.uni-kiel.de/confluence/display/KIELER/Ptolemy+Browser

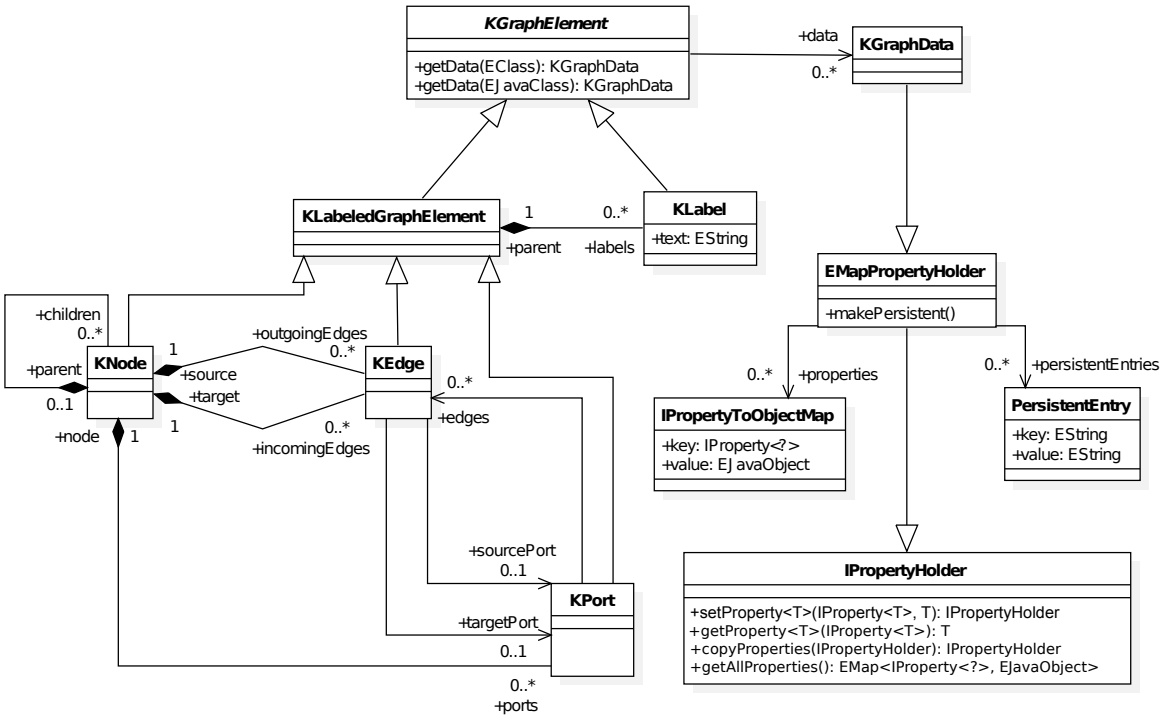


Figure 2.5. Meta model of the KGraph as defined in KIELER.

KGraph

In KIELER, the KGraph[5] is the central data structure when it comes to the representation of graphs. Its meta model is presented in Figure 2.5.

A KGraph has nodes that can be equipped with ports. Each graph contains a single node that does not belong to a parent node. This top-level node is necessary for the representation of the entire graph. In addition, nodes can contain other nodes to allow a hierarchical structure and can be connected by directed edges. Nodes as well as ports and edges can have an unlimited number of labels. Each KGraph element can store additional information such as data concerning the layout. Layout information comprises amongst other things the location, width, and height for nodes, ports, and labels or bend points for edges [SFvH09]. For several kinds of elements it also contains more unique data, for example a comment's text for a comment node. Such layout options are designed as pairs that contain a key and a value [SSM+13].

[5] http://rtsys.informatik.uni-kiel.de/confluence/display/KIELER/KGraph+Meta+Model

Transforming a Ptolemy Model to a KGraph

As mentioned in Section 2.2.4, KLighD sends the data from a Ptolemy model to its corresponding transformation to get a KGraph representation of the model. The synthesis of a KGraph from a Ptolemy model is very important for this thesis because the attachment of comments is implemented as its final stage.

The synthesis can be divided into three different parts: the transformation, the optimization, and the visualization.

Transformation The transformation performs the actual conversion from Ptolemy to KGraph elements for the majority of a diagram's elements. In this step, entities, relations, links, and ports are transformed.

First, every KNode receives attributes of its corresponding entity. This includes the marking of the KNode as a former Ptolemy element. Additionally, the type of the Ptolemy entity is saved in the KNode's annotations and the entity's properties are added to the KNode.

Once the entity itself has a KNode created for it, the entity's ports are identified. The identification of ports can be a difficult process because ports do not have to be explicitly defined in the MoML file. They often are included in the Java implementation of the entity they belong to. Thus, when trying to extract ports, the Ptolemy browser makes an attempt to instantiate the entity through the Ptolemy library to query it for its ports. Then, all the ports that are explicitly defined in the MoML file are identified as well. Subsequently, all ports found in the MoML file are added to their entity if they are not already registered in the library.

Each port should be marked as input, output, or both with the help of the definition from the Java library. However, if an entity does not have a library definition, it is not possible to distinguish the type for the ports.

Finally, for every entity, its child entities, relations, and links are added to the corresponding KNode. To achieve this, child entities and relations are transformed into KNodes and links into KEdges. At the end of this stage, KEdges have a source and target node and a source and target port. The problem is that KEdges in KIELER are directed, but links in Ptolemy do not provide any information concerning direction. Thus, at this stage of the transformation, the direction of KEdges is arbitrary. The task of finding the correct direction is accomplished in the optimization.

Optimization The optimization enhances the just created KGraph model. First, it is tried to deduce the direction of edges and the type of ports if their types have not already been determined. The direction of edges is inferred with the help of the ports whose types are already known. For example, if a port is an input port, the connected edge has to be incoming.

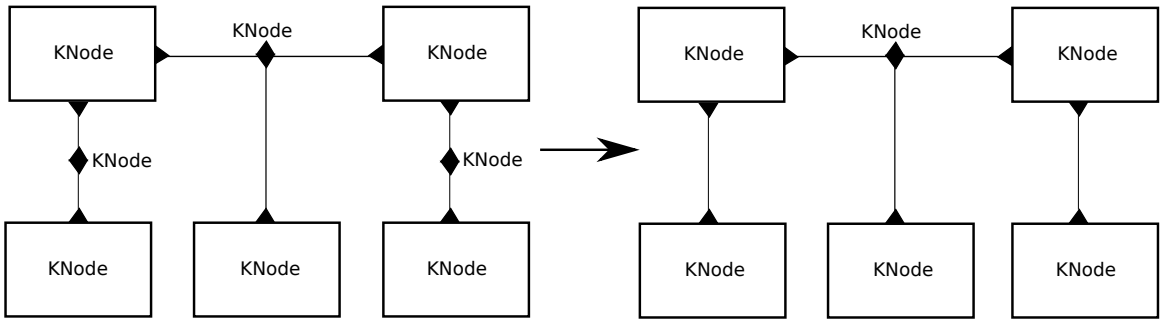


Figure 2.6. This diagram shows that KNodes that derive from relations can be deleted if they connect only two ports. KNodes that represent entities in Ptolemy are displayed as rectangles. KNodes that represent relations are displayed as rhombi.

Inferring unknown port types is handled in a very similar way: for every port that does not have a type the associated edge is regarded. If, for example, the edge is an incoming edge, the port is assumed to be an input port.

Second, the first stage of the transformation process has transformed every relation into a KNode. However, these nodes are not needed if the relation is part of a connection with exactly two ports. In this case, a simple edge is sufficient which simplifies the overall layout of the diagram. Thus, all unnecessary relations are removed. This step is shown in Figure 2.6.

Third, ports are removed from states of modal models because they are not needed due to the lack of data flow.

Fourth, certain properties are subsequently transformed into KNodes. In Ptolemy, some elements are represented as properties even though, in the diagram, they appear as entities. Examples for these properties are the director and a special title element. To be able to handle these elements properly in the layout algorithms, they are transformed into KNodes as well.

Another kind of property that is dealt with are parameters. All parameters belonging to a diagram are put into a list and this list is added to a single KNode.

Finally, two further properties are regarded to extract a diagram's comments. One of these properties is easy to access so that the comment can be retrieved without any problems. This can be seen in the code piece below. The property "text" can be found as well as its value "This is a comment.".

```
<property name="Annotation"
class="ptolemy.vergil.kernel.attributes.TextAttribute">
        <property name="textSize" class="ptolemy.data.expr.Parameter" value="10">
        </property>
        <property name="text" class="ptolemy.kernel.util.StringAttribute"
```

```
        value="This is a comment.">
        </property>
</property>
```

For the other property, the extraction of comments proves to be more difficult. This property contains the *configure* element with *svg-markup* mentioned in Section 2.2.2. The problem is that the parser that goes through the MoML file cannot handle *svg-markup* and discards it. Thus, the MoML file has to be searched specifically step by step for the property *_iconDescription*, the *configure* element, the *svg-markup* and the *text* element to get access to the comment.

```
<property name="annotation" class="ptolemy.kernel.util.Attribute">
        <property name="_iconDescription"
        class="ptolemy.kernel.util.SingletonConfigurableAttribute">
                <configure><svg><text x="20" y="20" style="font-size:10;
                font-family:Verdana; fill:black">
                This is a comment.</text></svg></configure>
  </property>
  </property>
</property>
```

Visualization The last part of the transformation process deals with the rendering of the model's elements. Depending on the type of the entity, different layout information are attached. Finally, the actual design of the entity is created. To that end, the KIELER Ptolemy library is searched for a rendering of the entity for the majority of entities. If it is not possible to find the entity in the library, it is displayed by a simple rectangle that has a black contour and a white background. Regarding the parameter node, all parameters of the list added to the node are displayed one below the other. For every parameter, a blue dot, the parameter's name and its value are shown.

Chapter 3

Heuristics

In this chapter, the heuristics implemented in this thesis are described. We start by explaining how title or author comments can be identified. Then, the computation of a comment's size is presented, followed by the computation of the alignment between a vertex and a comment. Finally, different approaches to finding the name of a non-comment vertex in a comment are explained.

3.1 Author Comments

In most of the demo diagrams from Ptolemy, there is a comment which identifies the diagram's author. An example of a diagram that contains an author comment is presented in Figure 3.1.

It is clear that an author comment should never be attached to a vertex because it provides general information about the diagram. Thus, by identifying a comment as an author comment, it can be removed from the list of attachable comments before the attachment process even starts to prevent wrong attachments.

The author comment displayed in Figure 3.1 is a perfect example of how these comments are composed in Ptolemy demo diagrams. Usually, the string "Author:" or "Authors:" is followed by the name(s) of the author(s). Of course, there are comments that do not conform to this rule. However, their number is very small.

The implementation is very simple. Every comment's text is converted to lowercase letters. Then, it is checked whether the comment contains the string "author".

3.2 Title Comments

Titles are important elements of texts because, when applied correctly, they communicate the key statement and raise interest [ES15]. In diagrams, titles seem to have the same function. Regarding the set of Ptolemy demo diagrams, several of them contain titles. Figure 3.1 provides an example of a title with the comment "Compared Clocks". Since title comments concern the entire diagram, just as author comments, they should not be attached to any vertices. Title comments are treated in the same way as author comments. It is tried to identify them to be able to delete titles from the list of attachable comments .

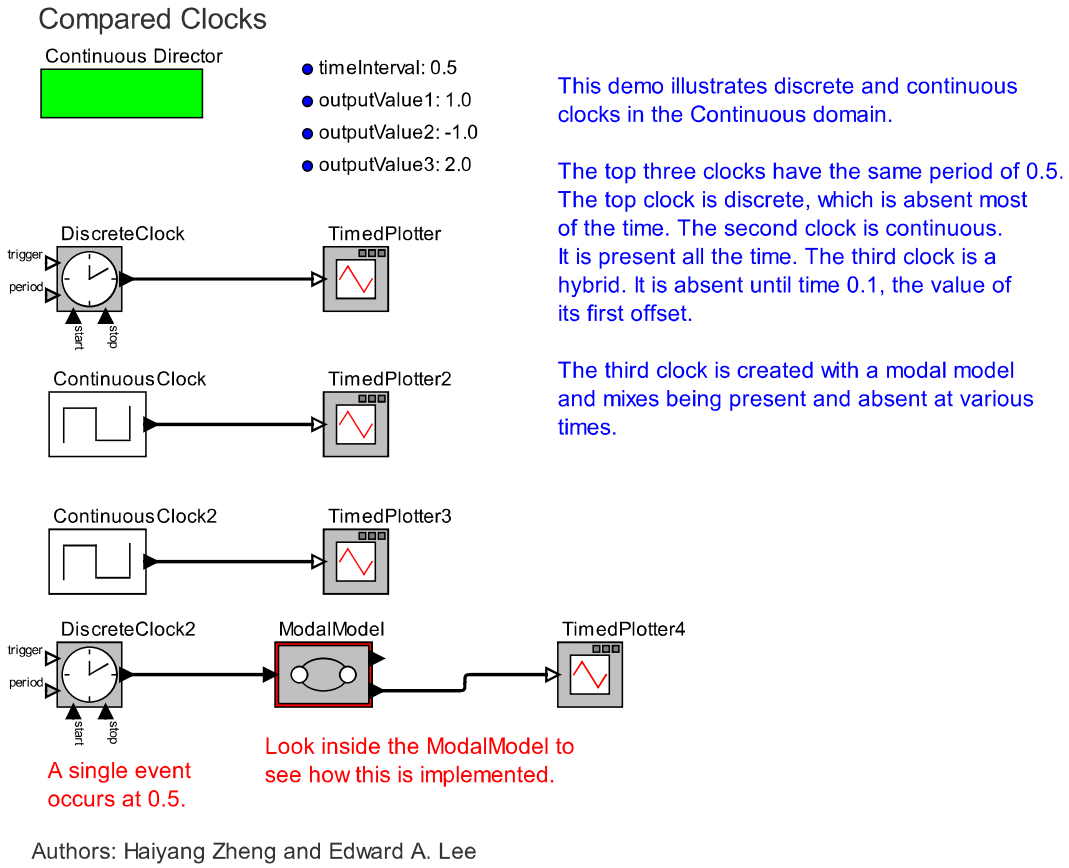


Figure 3.1. This diagram is taken from the set of demo diagrams from the Ptolemy tool. However, it was edited by adding a title for it to exhibit all characteristics of a diagram that lead to the heuristics implemented in this thesis.

In this thesis, two ways of detecting a title are presented. The first approach concerns the title property[1] provided by Ptolemy and already mentioned in Section 2.2.4. This property applies a large font size to the comment by default to automatically make it look like a caption.

Since this element is a property, it is transformed in the optimization phase of the transformation from Ptolemy model to KGraph. During this step, it is marked as a comment node, but also as a title node to be able to distinguish it from other comments. In the extraction phase, the node is added to the list of comment nodes. In order to prevent title comments from being attached, every comment node is checked whether it is marked as a title node at the beginning of the attachment process. If it is a title comment, no attachment is tried.

[1] http://ptolemy.eecs.berkeley.edu/ptolemyII/ptII10.0/ptII10.0.1/doc/codeDoc/index.html

The second approach uses the observation that titles often have larger font sizes than normal comments. The title comment in Figure 3.1 is an example for such a case. Note that the title comment is the only comment element that has the largest font size. This observation is important for the construction of the algorithm.

At first, the largest font size is determined. This algorithm consists of three steps:

1. For every comment, its font size is detected.
2. Every font size is inserted into a list. Font sizes that are larger than the first element of the list are added at the head of the list. Font sizes that have the same font size or are smaller than the first element of the list, are added at the list's tail.
3. After every font size is added to the list, the first element of the list is returned, if it is truly greater than the second element; otherwise, *0* is returned.

Because of the third step, it is ensured that the font size returned is truly greater than all the other font sizes. Finally, at the beginning of the attachment process, it is checked for every comment whether its font size is the font size determined by the algorithm above. If so, an attachment is not tried for this comment.

The question remains how the font size of a comment is detected. If its default settings have not been changed during the creation of the diagram, the font size is not mentioned in the MoML file. If it has been changed, there are two ways for the font size to be included in the MoML file. First, it can be a property that is inside the comment property. In this case, the font size can be extracted without problems by retrieving the value of the property. However, the font size can also be included in *svg-markup* in a *configure* element. In this case, the same procedure is applied as with comment texts in Section 2.2.4. If no font size can be found, the font size is set to its default value which is 14.

3.3 Size

Figure 3.1 shows that the size of comments varies. There are comments that concern the entire diagram and comments that relate to vertices. It is often found that comments that provide information regarding the entire diagram are larger than comments belonging to vertices. In Figure 3.1, a good example for a comment that gives general information is presented by the comment on the right that consists of three paragraphs of text. This comment is significantly larger than the comments relating to vertices.

This heuristic aims at differentiating between comments that concern the entire diagram and comments relating to vertices by calculating their sizes. To avoid attaching comments providing information about the whole diagram, a threshold, the *Maximum Comment Size*, is introduced. This threshold indicates the maximum allowed size to still be considered for attachments during the attachment process.

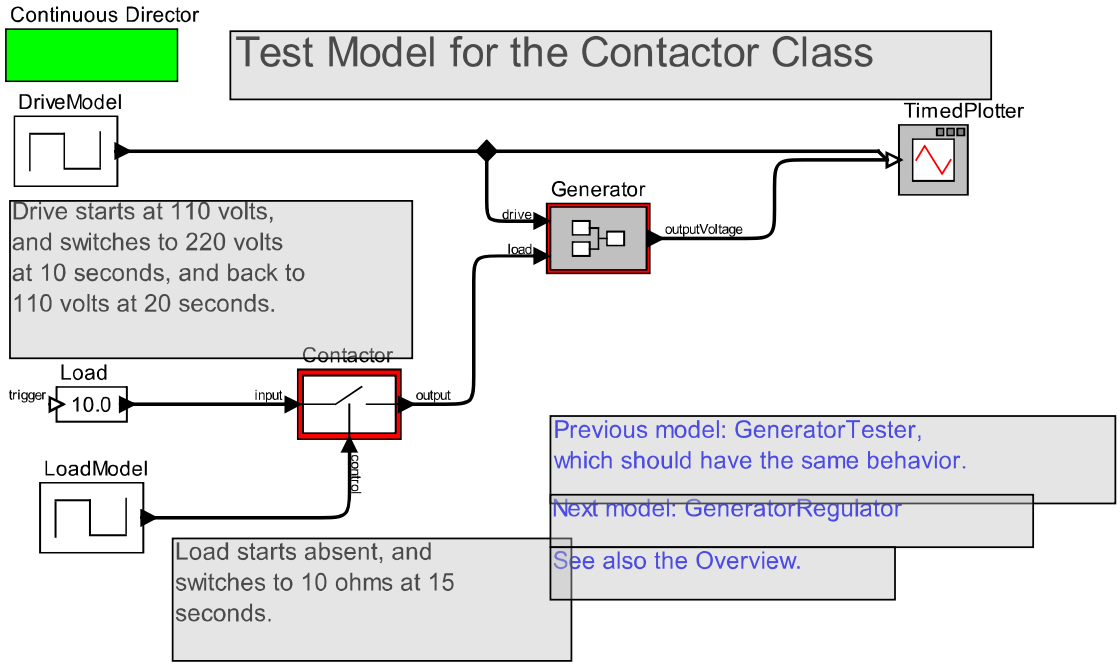


Figure 3.2. A demo diagram from Ptolemy in which the estimated sizes of the comments are shown as rectangles with gray background color.

For every comment, the size is calculated by multiplying its width with its height. The width and height of comments are estimated quantities because they are not deposited in the MoML file. For comments, KLighD approximates these quantities from the results of the comment's rendering during the transformation. Thus, the width and the height do not represent the width and height of the comments in Ptolemy, but the width and height of the comment's text after being rendered during the transformation. Figure 3.2 shows an example of a diagram where the actual sizes of comments differ significantly from the estimated sizes. The estimated sizes usually are larger than the original sizes. Especially the widths of the comments are bigger than in the original diagram.

Finally, it is checked whether the calculated size exceeds the maximum comment size. If the maximum comment size is exceeded, the comment is excluded from the attachment process.

3.4 Alignment

As mentioned in Section 1.2, Lidwell et al. explain that two aligned elements tend to be perceived as related [LHB03]. Figure 3.1 illustrates this statement: the comment "A single

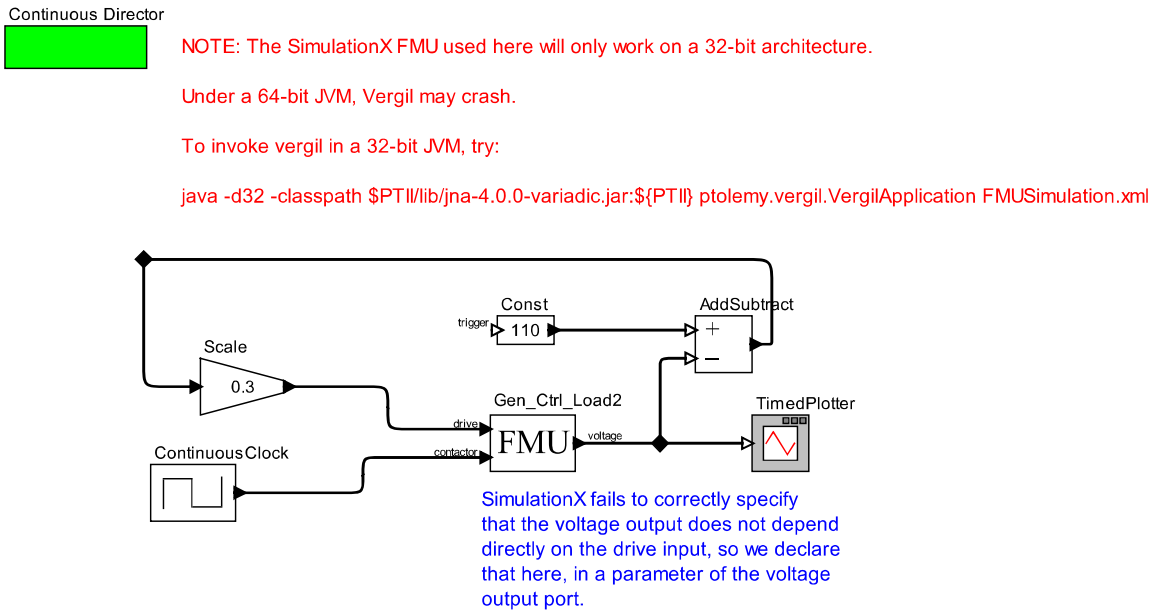


Figure 3.3. In this part of a Ptolemy demo diagram, it is clear that the comment at the very bottom belongs to the FMU element due to their alignment, even though regarding proximity, it could also be attached to *TimedPlotter*.

event occurs at 0.5." is perceived as belonging to *DiscreteClock2* due to the fact, amongst other things, that they are arranged along a straight vertical line. Additionally, when regarding the Ptolemy demo diagrams, it can be noticed that the proximity sometimes does not serve as a suitable heuristic. Figure 3.3 shows that it is possible that the arrangement along a straight line can be more successful.

For our purposes, elements can be arranged along either horizontal or vertical straight lines. An optimal horizontal alignment (that is, left-aligned or right-aligned) is obtained if either the left or the right side of both elements can be connected with a straight vertical line.

In Figure 3.4, two pairs of comments and vertices are displayed that show a perfect horizontal left and right alignment. However, developers usually do not arrange elements of a diagram in such a neat way: comments often protrude on the left or right side. This leads to the definition of horizontal alignment.

Definition 7. Let $G = (V, E, C)$ be a graph with comments. Let $c \in C$ be a comment and $v \in V$ be a vertex. We define the *horizontal alignment* of v and c as a function $a_h \colon V \times C \to \mathbb{R}$ that computes the minimum of $l, r \in \mathbb{R}$ with $l = |pos_x(v) - pos_x(c)|$ and $r = |pos_x(v) + width(v) - (pos_x(c) + width(c))|$.

As illustrated in Figure 3.5, the distance between the x-position of the comment and the

Figure 3.4. This figure shows two perfectly horizontally aligned pairs of vertices and comments.

Figure 3.5. This figure illustrates the definition of the horizontal alignment. The distance is calculated between the right sides or between the left sides of the comment and vertex.

x-position of the vertex is computed for the left alignment. The right alignment calculates the distance between the x-position plus the width of the comment and the x-position plus the width of the vertex, thus, the distance between the comment's right side and the verticis right side. Finally, the minimum of the left and right alignment is taken as the value for the horizontal alignment. In general, the following rule applies: the smaller the value of the calculated alignment, the better the alignment of the comment and the vertex.

An optimal vertical alignment is obtained if a straight horizontal line can be drawn along either the upper or the lower side of both elements. This is illustrated in Figure 3.6. Analogously to the horizontal alignment, an optimal arrangement often is is seldom found, leading to the definition of vertical alignment.

Definition 8. Let $G = (V, E, C)$ be a graph with comments. Let $c \in C$ be a comment and $v \in V$ be a vertex. We define the *vertical alignment* of v and c as a function $a_v \colon V \times C \to \mathbb{R}$ that computes the minimum of $t, b \in \mathbb{R}$ with $t = |pos_y(v) - pos_y(c)|$ and $b = |pos_y(v) + height(v) - (pos_y(c) + height(c))|$.

Figure 3.7 shows that for the upper sides, the distance between the y-position of the comment and the y-position of the vertex is calculated. Regarding the lower sides, the

Figure 3.6. This figure shows two perfectly arranged pairs of vertex and comment along a vertical line.

Figure 3.7. This figure illustrates the definition of the vertical alignment. The distance between the top sides or between the bottom sides of the vertex and comment are calculated.

distance between the y-position plus the height of the comment and the y-position plus the height of the vertex is calculated. Analogously to the horizontal alignment, the minimum of both distances is taken as the vertical alignment and the value for an optimal vertical alignment is zero.

Generally, there are several ways comments can be positioned in relation to vertices. As shown in Figure 3.8, a comment can be positioned above, below, or on one side of the vertex. Additionally, it can intersect the vertex and can be cater-cornered to the vertex. For different positioning possibilities, different alignment strategies are pursued.

First, we consider the case that a comment is positioned cater-cornered to a vertex, such as in Figure 3.1 the comment "A single event occurs at 0.5.", to *ModalModel*. In our opinion, the comment and vertex are not related regarding alignment in this scenario. Therefore, cater-cornered comments should not be included in the calculation of the alignment.

Second, it is defined that the horizontal alignment is computed if the comment is positioned below or above a vertex. It would also be possible to find out the vertical alignment for comments above or below vertices. However, vertical alignment cannot achieve better results than horizontal alignment in these cases without the comment overlapping the node in large parts.

Third, the vertical alignment is calculated if the comment is positioned on the left or right side of a vertex. Since for comments positioned on one side of a vertex, horizontal alignment is not able to yield better results than vertical alignment, it is not considered.

During the development of the algorithm, it was initially assumed that comments and vertices do not intersect because a comment normally should be placed next to a vertex.

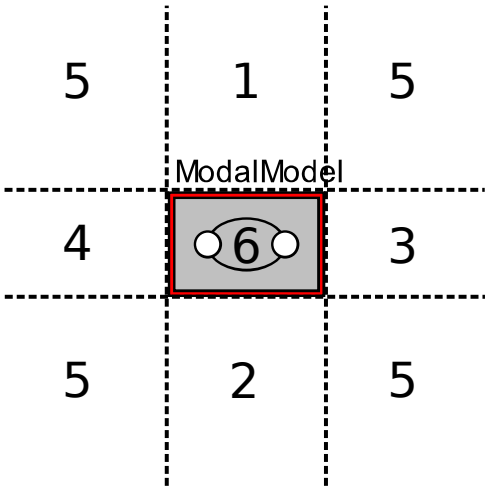


Figure 3.8. This figure illustrates the ways a comment can be positioned in relation to a vertex: 1 and 2) above or below the vertex, 3 and 4) on one side of the vertex, 5) cater-cornered to the vertex, 6) intersecting with the vertex. The modal model is taken from the Ptolemy tool.

However, since the estimation of a comment's size is inaccurate and developers often do not position comments precisely, intersections do occur. When coming across intersections, the horizontal as well as the vertical alignment is calculated and the minimum of these two values is chosen as the resulting alignment.

Now, we can finally define alignment.

Definition 9. Let $G = (V, E, C)$ be a graph with comments. Let $c \in C$ be a comment and $v \in V$ be a vertex. We define the *alignment* of v and c as a function $a\colon V \times C \to \mathbb{R}$, with

$$a(v,c) = \begin{cases} a_v(v,c), & (pos_y(c) + height(c) < pos_y(v) \vee \\ & pos_y(c) > pos_y(v) + height(v)) \wedge \\ & pos_x(c) + width(c) > pos_x(v) \wedge \\ & pos_x(c) < pos_x(v) + width(v) \\ a_h(v,c), & (pos_x(c) + width(c) < pos_x(v) \vee \\ & pos_x(c) > pos_x(v) + width(v)) \wedge \\ & pos_y(c) + height(c) > pos_y(v) \wedge \\ & pos_y(c) < pos_y(v) + height(v) \\ min(a_v(v,c), a_h(v,c)), & v \text{ intersects } c \\ \bot, & \text{otherwise} \end{cases}$$

Regarding the definition, the calculation of the vertical alignment is chosen if any part of a comment touches section 3 or 4 from Figure 3.8, but does not intersect with the vertex. The horizontal alignment is computed if a comment touches section 1 or 2 from Figure 3.8

and if there also is no intersection with the vertex. If comment and vertex intersect, the minimum of the vertical and horizontal alignment is taken as the alignment. Finally, if none of the cases above apply, meaning that the comment is cater-cornered to the vertex, no attachment is computed.

Algorithm 1 provides a detailed description of the algorithm. First, an alignment worse than acceptable is returned when the comment is cater-cornered to the vertex. Second, the alignment is calculated for a comment intersecting a vertex. Third, for a comment positioned above or below a vertex, the horiontal alignment is computed. Fourth, the vertical alignment is determined for a comment on the left or right side of a vertex.

Algorithm 1: Calculating the alignment between a vertex and a comment.

Data: Comment $c \in C$, Vertex $v \in V$
Result: Alignment between c and v

```
1  maximumAlignment = worst possible alignment
2  leftAlignment = |pos_x(v) − pos_x(c)|
3  rightAlignment = |pos_x(v) + width(v) − (pos_x(c) + width(c))|
4  topAlignment = |pos_y(v) − pos_y(c)|
5  bottomAlignment = |pos_y(v) + height(v) − (pos_y(c) + height(c))|
6  if c is cater-cornered to v then
7      return maximumAlignment + 1
8  else if c intersects v then
9      horizontalAlignment = min(leftAlignment, rightAlignment)
10     verticalAlignment = min(topAlignment, bottomAlignment)
11     return min(horizontalAlignment, verticalAlignment)
12 else
13     if c is above or below v then
14         return min(leftAlignment, rightAlignment)
15     else if c is left or right of v then
16         return min(topAlignment, bottomAlignment)
17     else
18         return maximumAlignment + 1
```

When applying the algorithm to models, diagrams such as the one in Figure 3.9 are produced. On the one hand, it shows the correct attachment of the comment relating to *Thermostat* in the original diagram. On the other hand, comments that concern the entire diagram are attached although they should not be, such as the author comment. The attachment occurs because even though the author comment and *Thermostat* are not close, their alignment is the best compared to the alignment with the other vertices. However, it is not probable that these elements should be connected because amongst other things the

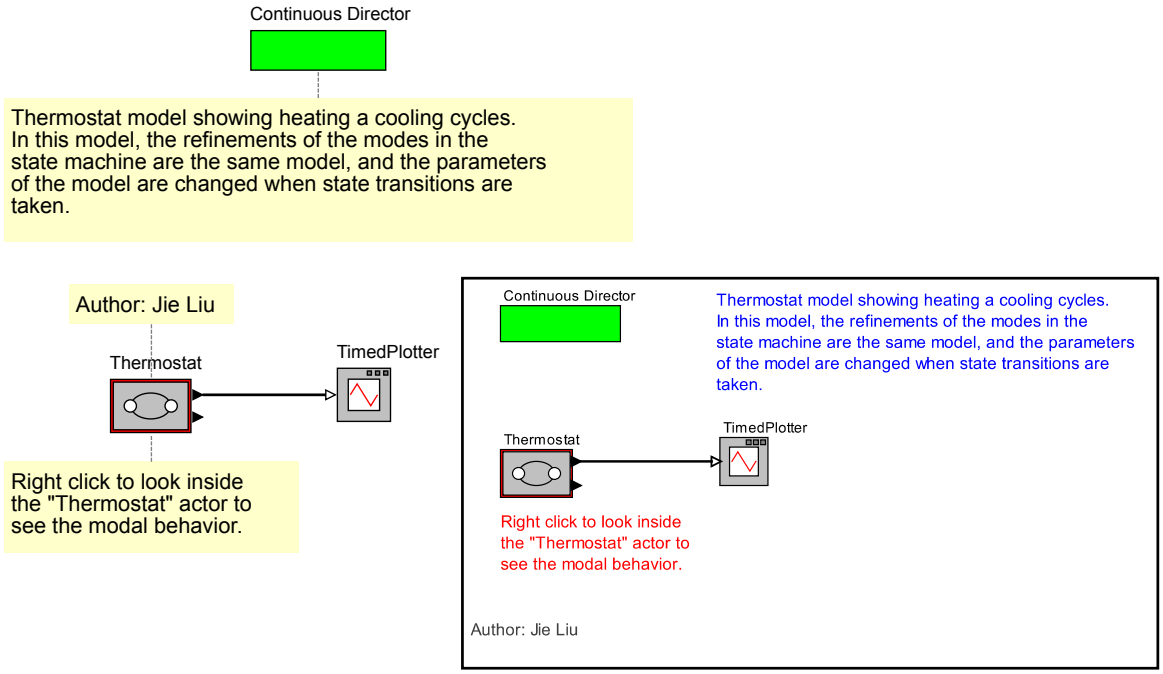


Figure 3.9. This figure contains two diagrams. The framed diagram is the original demo diagram from the Ptolemy tool. The second diagram is obtained after the automatic layout algorithm and the comment attachment algorithm with alignment as the attachment heuristic are applied.

distance between them is large. Therefore, the maximum attachment distance is included into the search for the normal vertex with the best alignment. No vertex can be attached to a comment if the distance between vertex and comment exceeds the maximum attachment distance even if they are well aligned.

3.5 Mentions of Actors

In several Ptolemy demo diagrams, it can be observed that comments relating to vertices contain the text of the vertices' labels. An exemplary diagram is shown in Figure 3.1, where the label of *ModalModel* appears in the comment that relates to it. However, many demo diagrams contain comments that provide general information about the diagram. In the text of these comments, labels of vertices are often mentioned to describe how the diagram's elements are associated. As mentioned several times above, it is desirable to attach a comment to a vertex if the comment relates to the vertex, but not if the comment concerns the entire diagram. Therefore, a comment should be attached to a vertex if the label of the vertex is the only label that appears in the comment.

In this thesis, two possibilities for detecting labels in comments are presented. For the

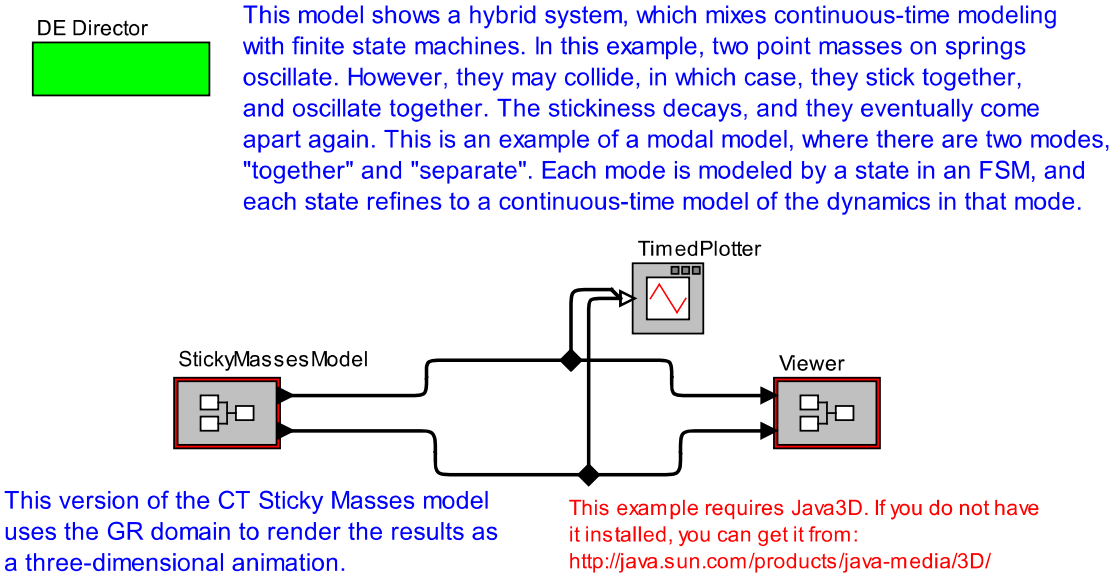


Figure 3.10. This Ptolemy demo diagram presents a comment that contains the words separated by space characters that form the label *StickyMassesModel*.

first approach, every comment is searched for unaltered label names.

The implementation of the first approach is simple. For a given comment it is checked if the label of any non-comment vertex appears in its text. If more than one label or no label is found, `null` is returned. Otherwise, the vertex whose label is found is returned.

However, many labels consist of more than one word. An example can be found in Figure 3.10: the label *StickyMassesModel* contains the words *sticky*, *masses*, and *model*. As can also be seen in Figure 3.10, in comments, these labels sometimes are written as separate words. Therefore, as a second approach, comments are searched for several versions of a label's text, the unaltered label name being among them. Additionally, if the label consists of more than one word, the comment is searched for these words separated by an arbitrary number of space characters. Moreover, the use of upper and lower case is not taken into account.

To be able to search for these words separated by space characters, a regular expression is constructed. A regular expression is a sequence of characters that represents a set of strings in which all elements have certain characteristics. Regular expressions have a particular syntax. The rules that affect the construction of the regular expression used in this thesis are explained below.

In this thesis, a regular expression is constructed for every label and this expression is matched with the comment. If the comment and the regular expression match because

the comment contains the label, and if no other label is found in the comment, the vertex whose label is found is returned.

When trying to construct a regular expression to find a label or a slightly altered version of the label in a comment, certain aspects have to be considered. First, a comment can contain an arbitrary number of arbitrary characters until the label is mentioned, and an arbitrary number of arbitrary characters can follow. This is illustrated in Figure 3.10, where the label *StickyMassesModel* is mentioned in the middle of the comment. Therefore, the regular expression has to start and end with `(.*)`. In this context, the dot stands for an arbitrary character and the star for an arbitrary number of repetitions, including zero. In between the `(.*)`, the label has to be represented.

When regarding the label, new words start before an upper case character. In addition, some labels contain numbers. In normal texts, new words and numbers usually are separated by space characters. Therefore, the regular expression contains `(\\s*)` before an upper case character and a number, where `\s` stands for a space character. However, if the label already contains a space character, no space characters are included.

Finally, the text of the comment and the text of the regular expression is converted to lower case letters. Thus, labels can be found without having to take into account the use of upper and lower case letters.

As an example, the regular expression for *StickyMassesModel* would look like this:

```
(.*)sticky(\\s*)masses(\\s*)model(.*).
```

Table 3.1 shows an excerpt of possible versions of the label *StickyMassesModel* as identified by the first and second approach. The first approach only finds the unaltered label, whereas the second approach identifies an arbitrary number of versions.

3.6 Putting It All Together

In this section, we try to combine the different heuristics in a useful way. To achieve this, we analyze every heuristic with a subset of Ptolemy demo diagrams and try to get clues from the analyses on how to combine the heuristics. In the next chapter, the evaluation, we will finally evaluate the most relevant combinations with another subset of Ptolemy demo diagrams. For this purpose, the set of Ptolemy demo diagrams was split into two complementary subsets. As mentioned above, the first subset was used to learn from the diagrams to be able to combine the heuristics optimally. The second subset is not used in this chapter, but in the next chapter to evaluate the most relevant combinations.

In this thesis, three combinations of heuristics are presented. Additionally, two combinations are extended by one of the approaches described in Section 3.5.

The subset used in this section consisted of 156 diagrams which contain 497 comments and meet our demands. Our requirements comprise that all diagrams have to contain

Table 3.1. This table shows which versions of the label *StickyMassesModel* are identified by the first and second approach. Since the second approach allows an arbitrary number of space characters between words, this approach identifies an arbitrary number of versions.

Version of the label	First approach	Second approach
StickyMassesModel	yes	yes
stickymassesmodel	no	yes
Stickymassesmodel	no	yes
stickyMassesmodel	no	yes
StickymassesModel	no	yes
Sticky Masses Model	no	yes
StickyMasses Model	no	yes
Sticky Masses model	no	yes
sticky masses model	no	yes
sticky masses model	no	yes

comments. Second, comments should not relate to any other elements than vertices. Third, comments should not belong to more than one vertex. Finally, it has to be possible to display the diagrams properly in Ptolemy, in the KIELER Ptolemy Browser and in a tool used to analyze the comment attachment.

The analysis of the heuristics and combinations was conducted analogously to the process described by Schulze and von Hanxleden [SvH14]. First, for all diagrams, comments were attached manually, which led to the definition of a reference attachment function. Then, one of the combinations of heuristics was applied to the diagrams, resulting in a heuristic attachment function. Finally, the heuristic attachment function was compared to the reference attachment function.

The comparison results in the number of *correct*, *changed*, *lost* and *spurious* attachments. An attachment is called correct if the heuristic attachment function yields the same result as the reference attachment function. An attachment is considered to be changed if the reference attachment function attaches the comment to a different vertex than the heuristic attachment function. Lost attachment means that a comment is attached by the reference attachment function, but not by the heuristic attachment function. Spurious attachments occur when a comment is not attached by the reference attachment function, but by the heuristic attachment function [SvH14].

As a first preparatory step, we reevaluated the proximity heuristic using the subset of demo diagrams to see whether we deal with a starting position comparable to the

results presented by Schulze and von Hanxleden. We achieved the results presented in Figure 3.11. In their evaluation, Schulze and von Hanxleden achieved success rates of up to 90%. However, success rates drop when the maximum attachment distance is increased. Additionally, changed attachments happen and the number of lost attachments is reduced. Most importantly, spurious attachments occur more often [SvH14]. When comparing the two evaluations, these progressions are very similar. However, Schulze and von Hanxleden generally obtained slightly better success rates. A reason for the different number of correct attachments could be that our evaluation included only a part of the diagrams Schulze and von Hanxleden used for their evaluation. Additionally, we were able to take into account new Ptolemy diagrams that were not available when their evaluation was conducted. Moreover, further elements of the diagram are considered in the attachment process which could increase the number of spurious attachments. Finally, texts of comments now are displayed with the font sizes defined in Ptolemy. Therefore, it is possible that a different comment size is estimated which leads to different values concerning the proximity.

To be able to compare the proximity heuristic with the alignment heuristic in Chapter 4, the comment attachment when using the alignment heuristic is analyzed. Figure 3.12 shows the results of this evaluation. It can be observed that the evaluation of the proximity heuristic and alignment heuristic provide almost identical results. Therefore, the alignment heuristic seems to be as valuable as the proximity heuristic regarding this subset of Ptolemy demo diagrams.

As another preparatory step, we analyze the heuristics that search for the unaltered label or the label in which space characters are included. Figure 3.13 visualizes the results of the evaluation. Differently than expected, the search for the unaltered label yields better results than the second more liberal approach. Due to the same number of lost attachments for both approaches, it is a wrong assumption regarding this subset of Ptolemy demo diagrams that comments that belong to a vertex preferably contain the words of the verticis label separated by space characters if the label consists of more than one word. Searching for different versions of the label rather allows attachments to vertices that are not related to the comment, shown in the evaluation by the elevated number of spurious attachments. Therefore, for further purposes, we will only use the heuristic that searches for the unaltered labels.

As the final step of the preparation, the maximum comment size has to be determined: the comment size above which the comment is excluded from the attachment process. First, we calculated the average comment size, resulting in 41602.35, to get a first clue as to which size would be appropriate. Then, we applied the proximity heuristic with different maximum comment sizes to the diagrams. The results are presented in Figure 3.14. Since a maximum comment size of 20000 yields the best results, this maximum comment size is used in all further combinations of heuristics.

The first approach presented in this thesis combines the heuristics that exclude comments from being attached with the proximity heuristic. The idea for this approach

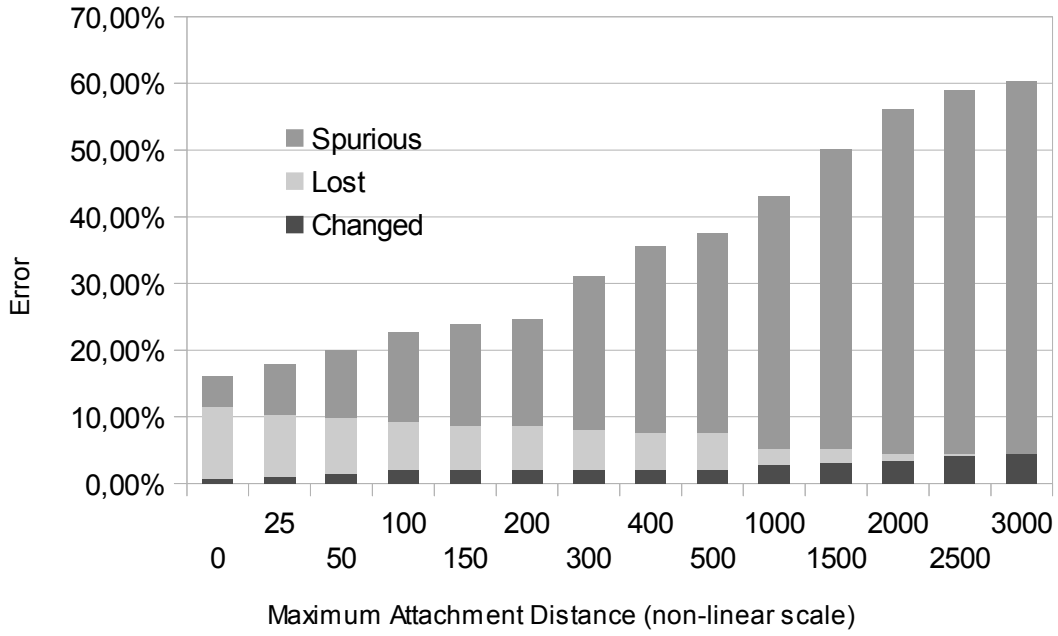


Figure 3.11. Error rate of the comment attachment when applying the proximity heuristic and different maximum attachment distances. Best success rates were achieved when applying low maximum attachment distances. When the maximum attachment distance increases, the number of correct attachment decreases. In addition, the number of lost attachments decreases, whereas the number of spurious attachments increases significantly. The number of changed attachments increases only slightly.

developed after regarding the evaluation of the proximity heuristic from Schulze and von Hanxleden [SvH14]. In their evaluation, even at small maximum attachment distances, spurious attachments occur. The number of these attachments increases with the maximum attachment distance. Therefore, it is proposed that spurious attachments can be avoided by adding the heuristics that exclude comments from being attached. Thus, we suggest to use the proximity as the main heuristic and add the heuristics that exclude title, author comments and comments that exceed a certain size.

Figure 3.15 presents the results of the evaluation of our first approach. Success rates of up to 89% are achieved.

The second approach is very similar to the first one because it combines the heuristics that exclude comments from the attachment process with the alignment heuristic. The motivation for this approach is the same as the motivation for the first approach. Figure 3.16 demonstrates the results of the evaluation. Both heuristics achieve very similar results.

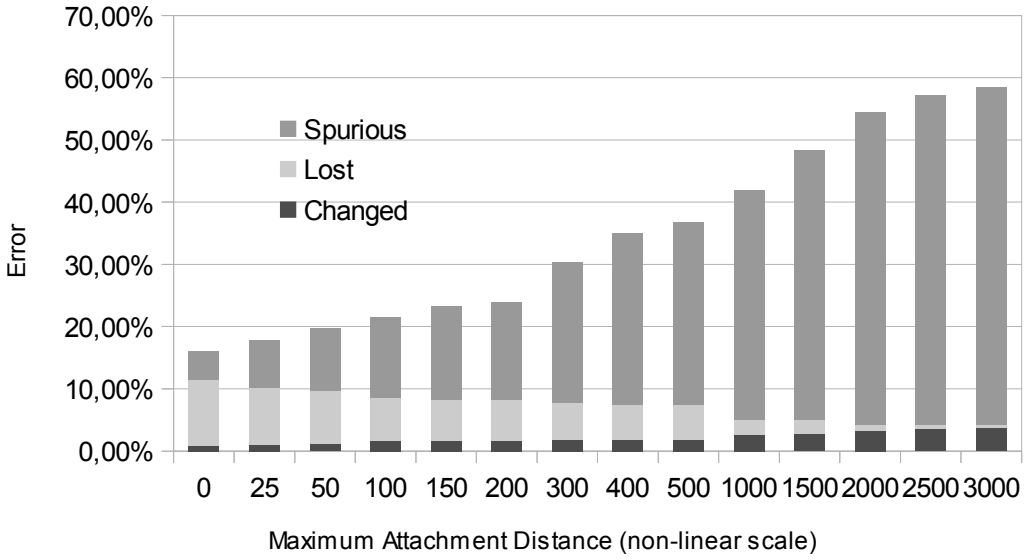


Figure 3.12. This diagram shows the error rate of the comment attachment when applying the alignment heuristic and different maximum attachment distances. The results are very similar to the proximity-based results in Figure 3.11.

The best success rate is obtained when applying the alignment heuristic with a maximum attachment distance of 50 together with the heuristics that exclude comments from being attached.

Another way to achieve a better comment attachment would be to lower the number of lost attachments. We propose that the heuristic that searches the comment for the unaltered label achieves this because it also identifies comments that are not in the vicinity of the vertex. Therefore, we add the label heuristic to the first and second approach. Both approaches now handle the comment attachment in the following way. First, the title, author and size heuristics deplete the pool of attachable comments. Then, the label heuristic is applied. If exactly one label can be found in a comment, the label's vertex is attached. Otherwise, the proximity or alignment heuristic capped by the maximum attachment distance is used.

Figure 3.17 and Figure 3.18 show the results of the evaluations of both extended approaches. They again yield very similar results. It can be observed for both evaluations that the success rates do not improve, but stay the same.

Finally, the third approach is presented. It assumes that the proximity and the alignment are complementary concepts. For some pairs of comment and vertex, the proximity yields

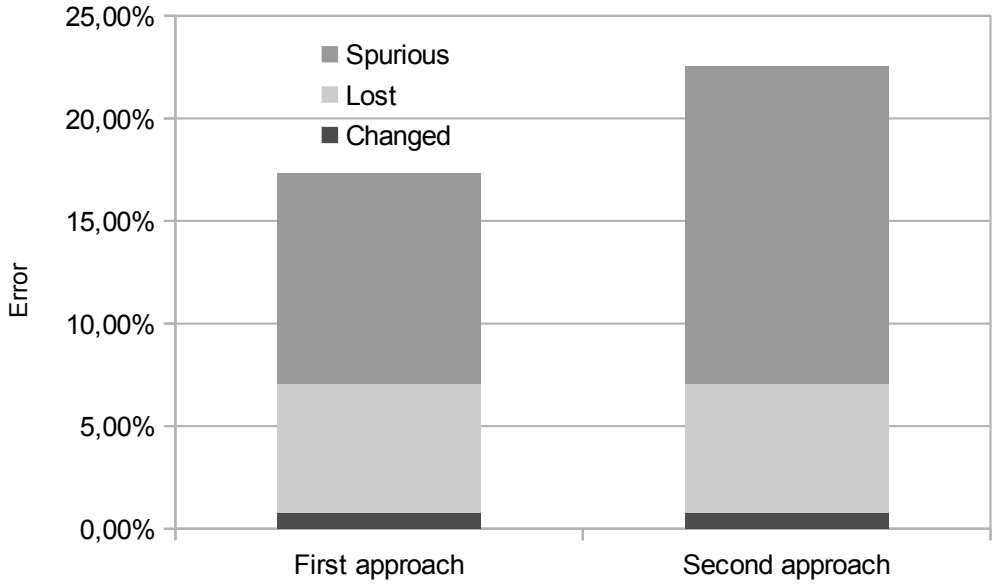


Figure 3.13. Error rate of the comment attachment when applying the heuristics that search for the unaltered label (first approach) or the label in which space characters are included (second approach). The evaluation of both heuristics provides the same number of changed and lost attachments. However, the number of spurious attachments is higher if the label with included space characters is used.

better results than the alignment and vice versa. As already mentioned in Section 3.4, Figure 3.3 provides an example of a comment for which only the alignment heuristic achieves a correct attachment. Thus, we propose a weight function $w\colon V \times C \to \mathbb{R}$ that takes into account the proximity and the alignment. It is $w = e \times p(v,c) + f \times a(v,c)$ with $(e,f) \in \{(1,0.8),(1,0.6),(1,0.4),(1,0.2),(0.8,1),(0.6,1),(0.4,1),(0.2,1)$ and $p\colon V \times C \to \mathbb{R}$ as the proximity. Remember that $a\colon V \times C \to \mathbb{R}$ is the alignment heuristic. Since for the first and second approach 50 proved to be a very good maximum attachment distance, this value is used. Additionally, the heuristics that exclude comments from the attachment process are applied. However, since the proximity heuristic and the alignment heuristic achieve very similar results, we only expect the outcome to differ slightly from the first and second approach.

Figure 3.19 presents the results of the evaluation of the weight function. As expected, the success rates do not differ from the best success rates achieved by the first and second approach. Additionally, the weights do not have any effect on the outcome.

Thus, the combination of heuristics we propose to use for the evaluation is the alignment

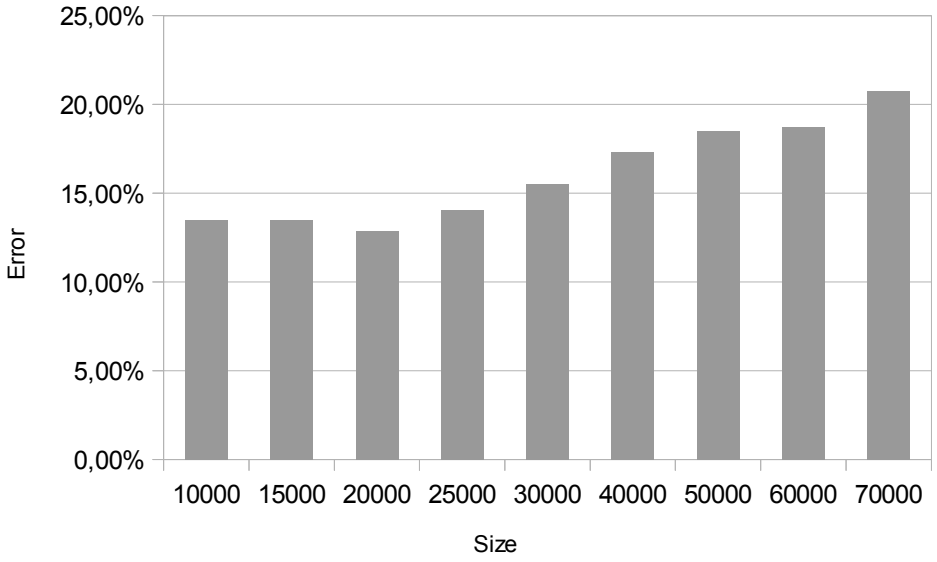


Figure 3.14. Error rate of the comment attachment when applying the proximity heuristic with maximum attachment distance = 150 and different maximum comment sizes. The best success rates are obtained when using a maximum comment size of 20000.

heuristic capped by a maximum attachment distance of 50 combined with the heuristics that exclude comments from being attached. Since we are also interested in the question whether the label heuristic can lower the number of lost attachments, we will also evaluate the alignment heuristic with maximum attachment distance of 25 together with the heuristics that remove comments from the attachment process and the label heuristic.

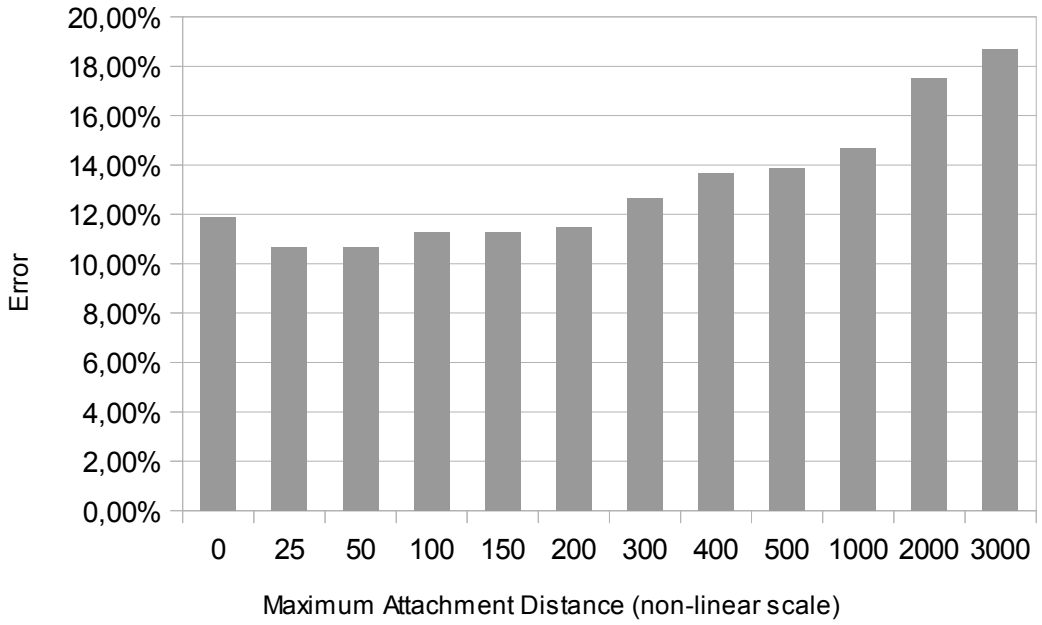


Figure 3.15. Error rate of the comment attachment when applying the proximity heuristic and the heuristics that exclude the title, author comments, and comments exceeding a size of 20000. The best success rates are obtained when using a maximum attachment distance of 25 or 50.

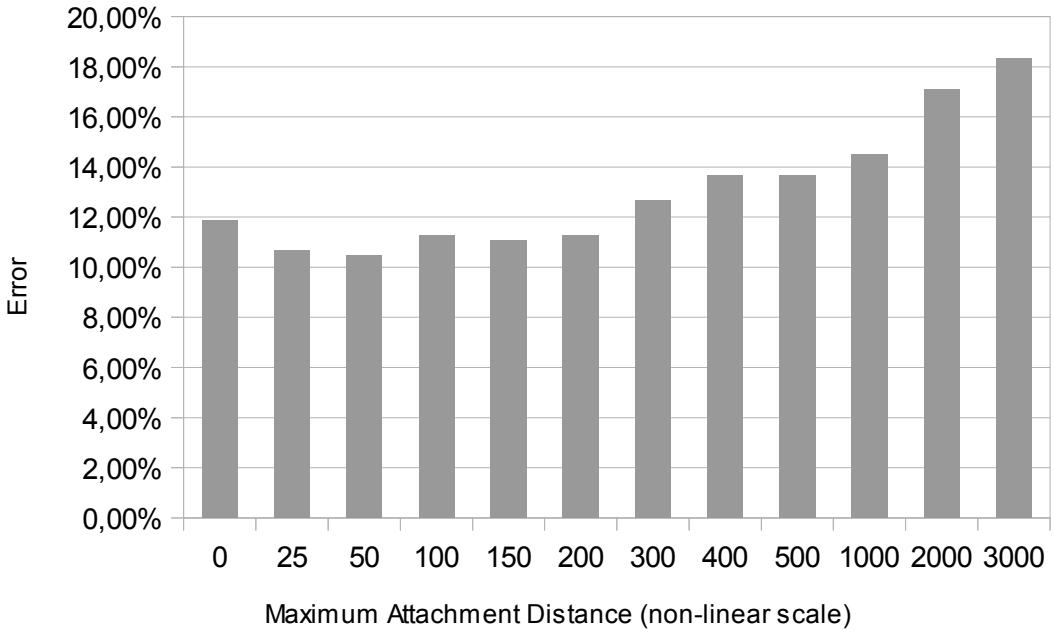


Figure 3.16. Error rate of the comment attachment when applying the alignment heuristics and the heuristics that exclude the title, author comments, and comments exceeding a size of 20000. The best success rates are obtained when using a maximum attachment distance of 50.

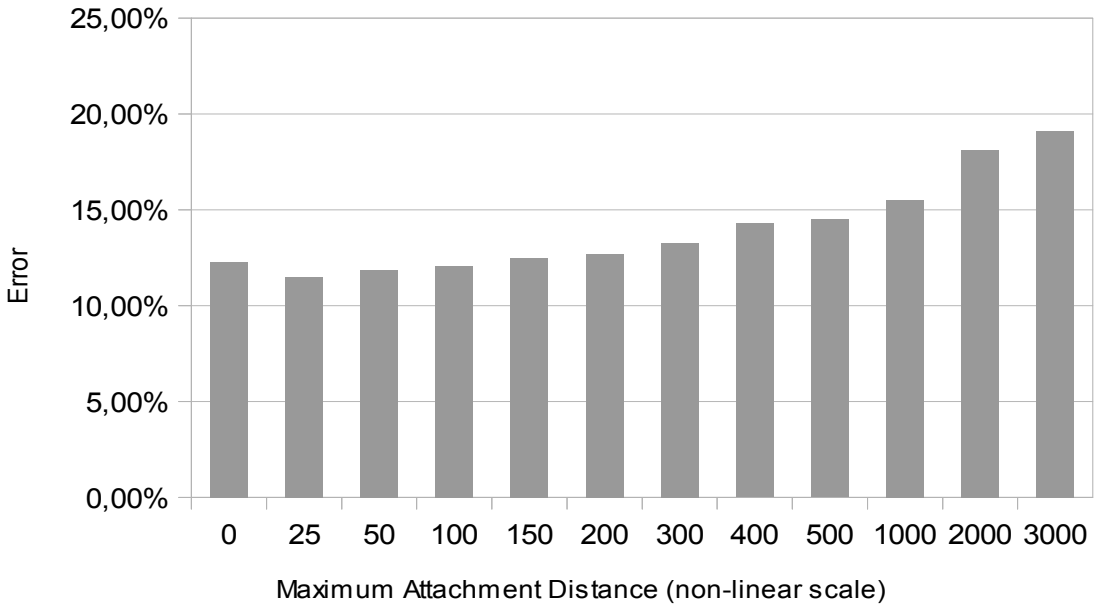


Figure 3.17. Error rate of the comment attachment when applying the proximity heuristics, the heuristics that exclude the title, author comments, and comments exceeding a size of 20000 and the label heuristic. The best success rates are obtained when using a maximum attachment distance of 25.

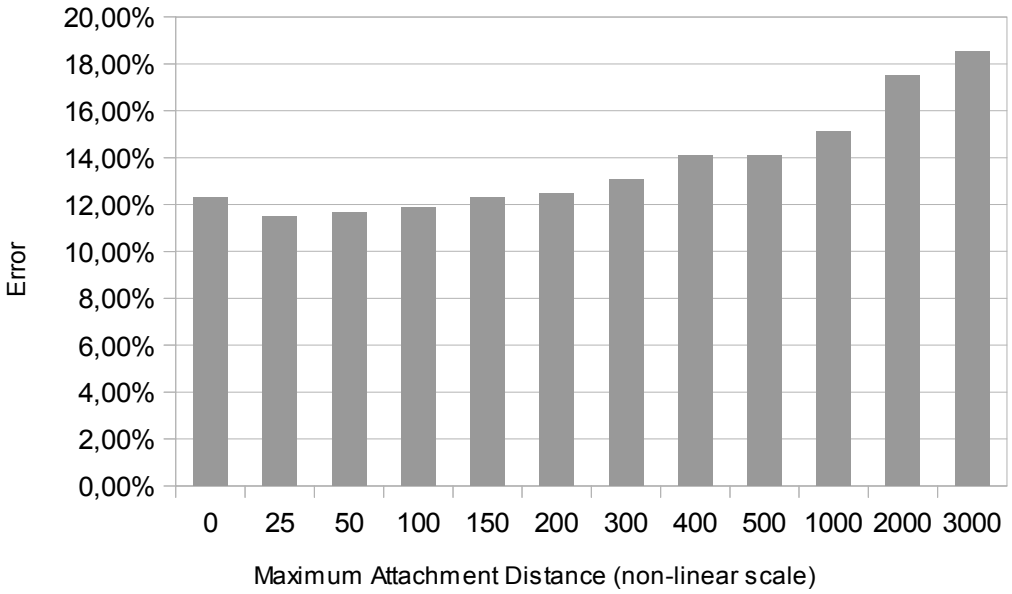


Figure 3.18. Error rate of the comment attachment when applying the alignment heuristics, the heuristics that exclude the title, author comments, and comments exceeding a size of 20000 and the label heuristic. The best success rates are obtained when using a maximum attachment distance of 25.

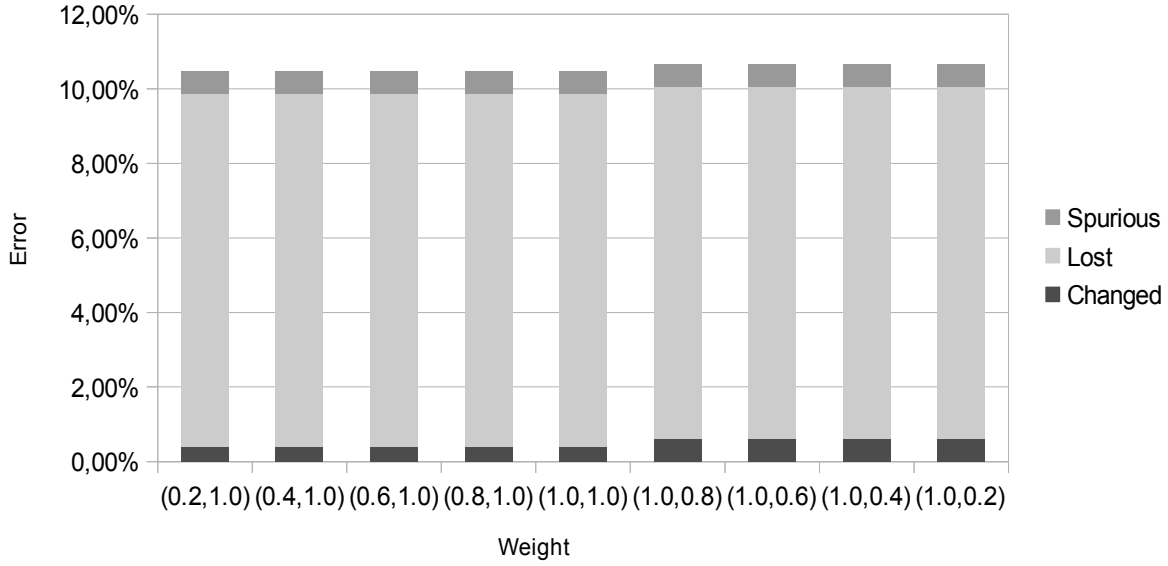


Figure 3.19. Error rate of the comment attachment when applying a weight function as well as the heuristics that exclude the title, author comments and comments that exceed a certain size. The maximum attachment distance is set to 50. Success rates are constant regardless of the weights used. (e, f) with $e, f \in \{(1, 0.8), (1, 0.6), (1, 0.4), (1, 0.2), (0.8, 1), (0.6, 1), (0.4, 1), (0.2, 1)$ represents the prefactors. e is the prefactor of the proximity heuristic, f the prefactor of the alignment heuristic. At (1,1), both heuristics have the same weight. By decreasing one prefactor, the weight of the respective heuristic is reduced.

Chapter 4

Evaluation

In this chapter, the results of the evaluation of the two combinations of heuristics proposed in Section 3.6 are discussed. The first combination of heuristics consists of the alignment heuristic capped by a maximum attachment distance of 50 and the heuristics that exclude comments from the attachment process. The second combination comprises the alignment heuristic with a maximum attachment distance of 25, the heuristics that exclude comments and the heuristic that searches for the unaltered label in a comment's text.

For the evaluation, a subset of the Ptolemy demo diagrams different from the subset used in Section 3.6 was taken. This subset consisted of 157 diagrams which contain 483 comments and meet the requirements. Every diagram was required to contain comments. Additionally, comments were only allowed to relate to vertices and should not relate to more than one vertex. Moreover, diagrams were excluded if comments were explicitly attached to vertices. Finally, it was necessary to be able to display the diagrams properly in Ptolemy, in the KIELER Ptolemy Browser, and in a tool used to analyze the comment attachment.

The goal of this evaluation was to find out whether our combinations of heuristics improve the success rates of automatic comment attachment compared to the proximity heuristic introduced by Schulze and von Hanxleden [SvH14]. Additionally, we wanted to estimate the practical use of our combinations of heuristics so that we compared the error rates of our combinations of heuristics with the error rates of the heuristic where no comment attachment is applied.

The evaluation was conducted in the same way as described by Schulze and von Hanxleden [SvH14]. As a preparatory step, a reference attachment function was defined by attaching the comments manually for all diagrams in the subset. Second, for every approach, that is to say for the first combination of heuristics, for the second combination of heuristics, for the proximity heuristic and for the no attachment heuristic, a heuristic attachment function was introduced by applying the respective heuristic to the diagrams. Finally, for every approach, the heuristic attachment function was compared to the reference attachment function.

The comparison produces four criteria that describe the quality of the comment attachment. These criteria are the number of correct, changed, lost and spurious attachments. An attachment is considered to be correct if the heuristic attachment function yields the same results as the reference attachment function. Changed attachments occur when a

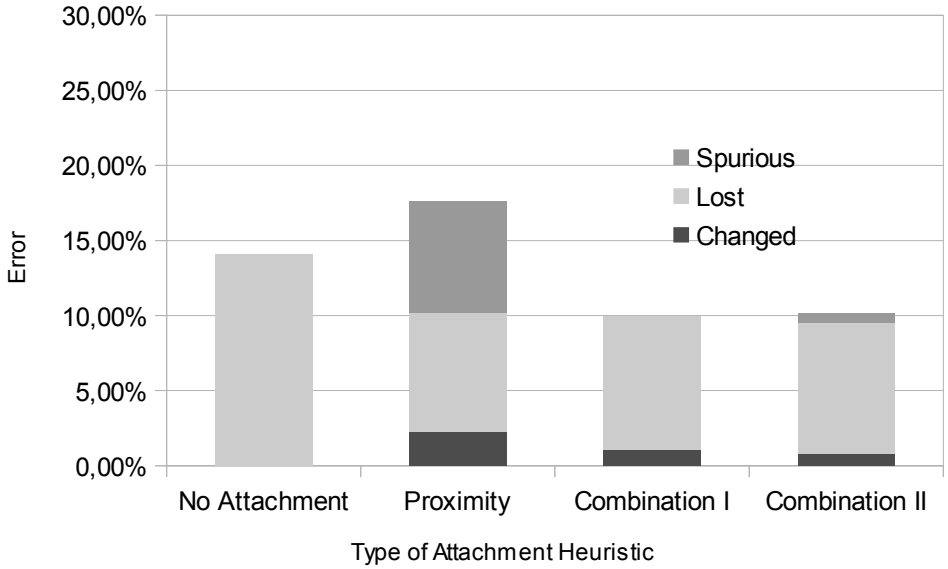


Figure 4.1. This diagram shows the error rate of the comment attachment when applying the proximity heuristic with a maximum attachment distance of 25 or one of the two combinations or no attachment. Combination I stands for the alignment heuristic capped by a maximum attachment distance of 50 plus the heuristics that exclude comments. Combination II means that the alignment heuristic with a maximum attachment distance of 25 is joined with the excluding heuristics and the label heuristic.

comment is attached to a different vertex in the heuristic attachment function than in the reference attachment function. Lost attachment means that a comment is attached in the reference attachment function, but not in the heuristic attachment function. An attachment is spurious if a comment is attached by the heuristic attachment function, but not by the reference attachment function [SvH14].

Figure 4.1 shows the results of the evaluation of the combinations, the proximity heuristic, and the case where no attachment is applied. First, it can be observed that the success rates of the combinations are slightly higher than the success rate of the case where no attachment is applied regarding this subset of Ptolemy demo diagrams. Additionally, the rate of correct attachments of the proximity heuristic is less successful than the rates of the combinations.

In Section 3.6, it was assumed that the heuristics that exclude comments from the attachment process help to decrease the number of spurious attachments. In Figure 4.1, it can be observed that the percentage of spurious attachments falls from 7.65% when applying the proximity heuristic to 0.6% when using the first combination. This supports the statement that excluding heuristics lowers the number of spurious attachments. Another

reason for the drop of spurious attachments could be that different main heuristics were used. However, as described in Section 3.6, the proximity and alignment heuristic yield very similar results.

A further assumption in Section 3.6 was that the heuristic that searches for the unaltered label can decrease the number of lost attachments. However, when we look at the results for both combinations in Figure 4.1, the number of lost attachments is not reduced when applying combination II, which contains the heuristic that searches for an unaltered label in contrast to combination I, to the demo diagrams. The application of this combination of heuristics rather leads to a slight increase in the number of spurious attachments.

Second, we compared the different attachment heuristics by using only the diagrams in which comments are attached to vertices. This set consists of 35 diagrams that contain a total of 166 comments. The results of the evaluation are presented in Figure 4.2. It can be observed that the success rates of the combinations are higher than the success rate of the proximity heuristic due to different numbers of spurious attachments. It is confirmed that the number of lost attachments is not affected by the application of the label heuristic.

To sum up, it appears that a combination of heuristics provides better success rates than the proximity heuristic alone. We are able to decrease the number of spurious attachments, but cannot alter the number of lost attachments for the respective maximum attachment distance. When comparing our approach to the case where no automatic comment attachment is applied, it yields slightly better results. However, since the success rates of the combinations of heuristics are close to the success rates of the no attachment heuristic, further approaches regarding automatic comment attachment could be considered.

An explanation for the very similar success rates to when no automatic comment attachment is applied could be that this subset of Ptolemy diagrams only has few diagrams in which comments should be attached. Out of 483 comments, only 166 are related to a vertex. Therefore, the no attachment heuristic is strong regarding this pool of diagrams. However, in a set of comments where a lot of comments are attached to vertices, the no attachment heuristic would become less important.

One further reason for the fact that we were not able to achieve better results could be the estimation of the shapes of the comments. This leads to a distorted representation of the diagrams which can cause incorrect attachments.

The lack of better success rates could also be caused by the large number of possible combinations of the heuristics. It is possible to change the maximum comment size and the maximum attachment distance. Additionally, one can activate or deactivate the label heuristic. Therefore, we may not have found the optimal combination of heuristics yet.

Another reason could be that developers sometimes create diagrams that do not follow general layout rules. This makes it harder for the heuristics to recognize relations.

Moreover, we found that the proximity and alignment heuristic do not consider the case where only the borders of a comment and a vertex touch. In this case, the comment and the vertex do not intersect, but the comment is also not positioned on one side of the

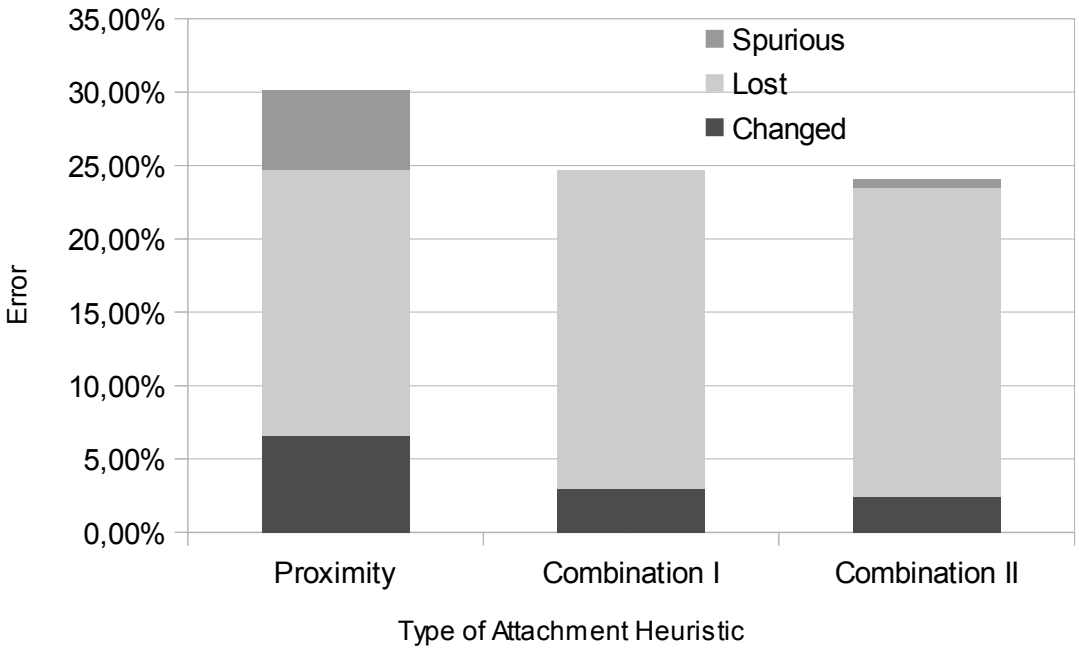


Figure 4.2. For this figure, only diagrams in which comments are attached to vertices are taken into account. It shows the error rate of the comment attachment when applying the proximity heuristic or one of the two combinations or no attachment. Combination I stands for the alignment heuristic capped by a maximum attachment distance of 50 plus the heuristics that exclude comments. Combination II means that the alignment heuristic with a maximum attachment distance of 25 is joined with the excluding heuristics and the label heuristic.

vertex. Therefore, this comment is excluded from the attachment process. If this scenario happens, a lost or changed attachment occurs. However, we believe that this situation arises only rarely so that the results of the evaluation are not altered considerably.

A limiting factor of this evaluation consists in the fact that all diagrams examined in this thesis are part of a set of demo diagrams developed by one research group. When regarding the diagrams, it seems that certain layout guidelines are established by the group. The uniform construction of the author comment and its positioning at the bottom left corner is one example for such a guideline. Additionally, the number of authors is limited to the members of the research group. Therefore, it could be easier to find layout patterns in this set of diagrams than in a set of arbitrary diagrams. Moreover, some heuristics like the author heuristic could be unique to this set of diagrams because other developers follow different layout guidelines.

Chapter 5

Conclusion

The goal of this thesis was to improve the success rate of automatic comment attachment for a set of demo diagrams from the Ptolemy tool. To achieve this, an already existing comment attachment algorithm was to be extended by implementing additional heuristics. These heuristics include the alignment of comments, the exclusion of the title, of the author comment, as well as of comments that exceed a certain comment size, and the mentioning of labels in comments. For the mentioning of labels, two approaches were presented. The first approach searches for the unaltered label, whereas the second approach tries to find several versions of the label's text.

We then evaluated the applicability of every heuristic with a subset of the demo diagrams from the Ptolemy tool. This led us to the development of different kinds of combinations of heuristics. Two combinations of heuristics were selected for the evaluation. The first combination uses the alignment heuristic capped by a maximum attachment distance of 50 and the heuristics that exclude comments from the attachment process. The second combination consists of the alignment heuristic with a maximum attachment distance of 25, the heuristics that exclude comments, and the label heuristic.

In the evaluation, the combinations were compared to the proximity heuristic in isolation and to the case where no comment attachment is applied. To be able to compare the different approaches, a second subset of the demo diagrams from Ptolemy which differed from the first one was used. Both combinations yield better success rates than the proximity heuristic. The reason for this is that the heuristics that exclude comments from the comment attachment lower the number of spurious attachments considerably. It was assumed that the second combination could lower the number of lost attachments because the label heuristic also finds related vertices that are not in the vicinity of the comment. However, the number of lost attachments is not affected by the label heuristic. Most importantly, the combinations of heuristics only yield slightly better results than if the comment attachment heuristic is not applied. Therefore, further work has to be done to be able to attach comments automatically.

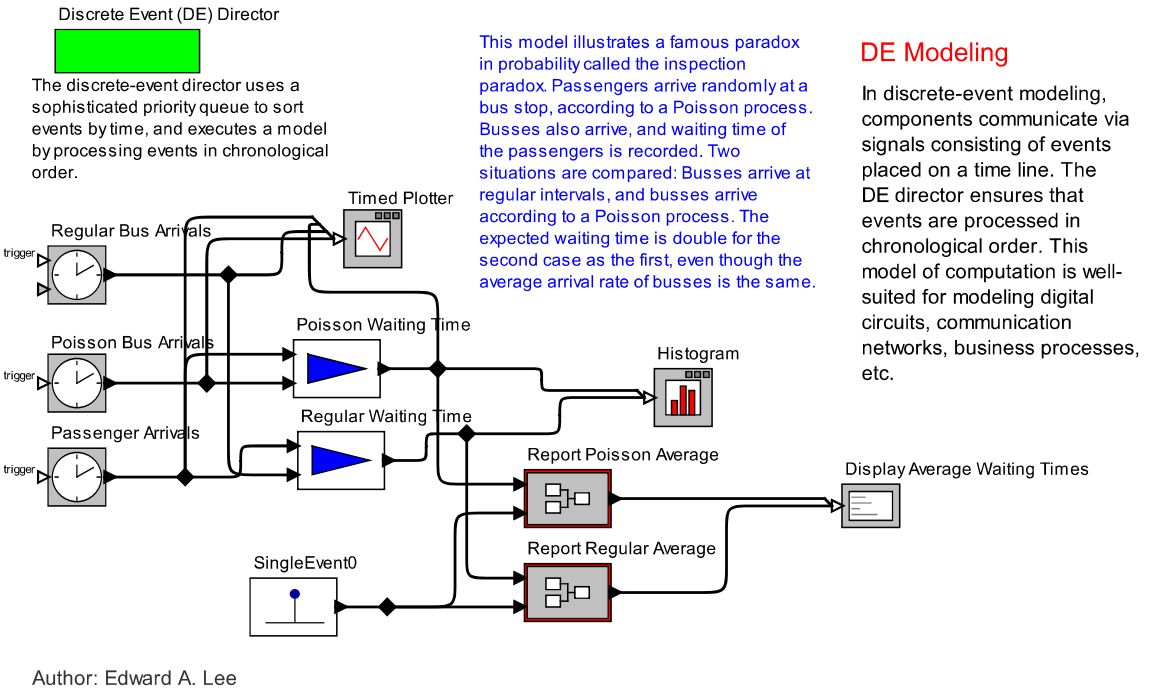


Figure 5.1. In this Ptolemy demo diagram, the comment "DE Modeling" and the comment directly below are related. "DE Modeling" serves as the title for the comment below.

5.1 Future Work

This thesis presents intuitive combinations of heuristics. Since these combinations only achieve slightly better results than if no attachment is applied, also further approaches regarding automatic comment attachment should be considered. For example, an algorithm based on artificial learning could be of use.

Additionally, in this thesis, only diagrams that do not attach comments to other elements than vertices are considered. However, comments can be attached to all kinds of elements. Therefore, comment attachment to edges, ports, or parameters could be realized. Moreover, comments can relate to other comments, as can be seen in Figure 5.1. Thus, one could also try to realize a comment attachment to comments.

One very important aspect of this thesis is that all diagrams examined are part of a set of diagrams created by one research group. Therefore, the diagrams are often designed in a similar way which could facilitate the application of heuristics. Figure 5.2 shows a diagram created during our own studies. At first sight, the diagram looks quite similar to the Ptolemy demo diagrams, which could be explained by the fact that the model is based on a model from a book written by members of the research group that developed

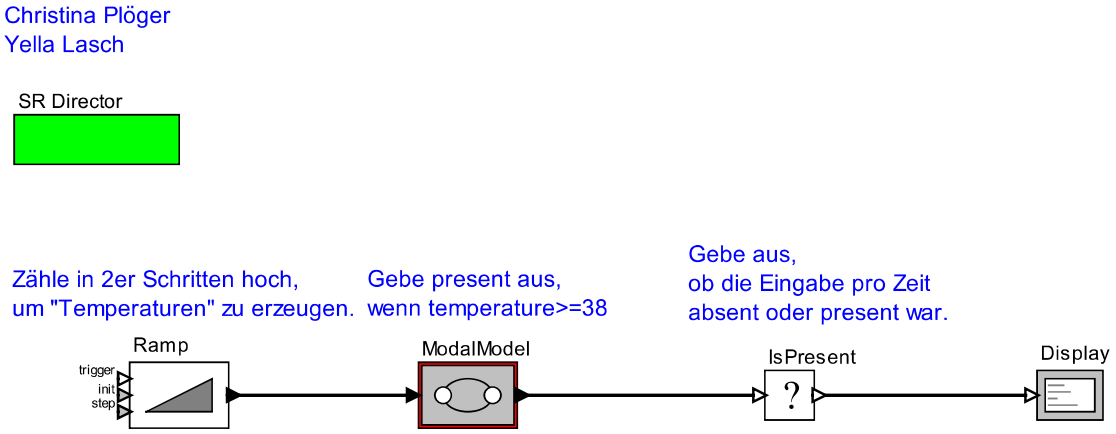


Figure 5.2. An event counter based on a model by Lee and Seshia [LS11, sec. 3.1].

Ptolemy. However, it would not be possible to identify the author comment with the heuristics presented in this thesis. Therefore, it would be very interesting to see the success rates of the heuristics when being applied to diagrams from different authors or even from different tools. It could be tried to refine the heuristics so that they are applicable to different kinds of diagrams.

List of Figures

Appendix A

Abbreviations

KIELER Kiel Integrated Environment for Layout Eclipse Rich Client
A framework for automatic layout.

MoML Modeling Markup Language
An XML-based file format.

KLighD KIELER Lightweight Diagrams
A framework that coordinates the generation and display of diagrams.

KIML KIELER Infrastructure for Meta Layout
A framework that deals with the choice of suitable layout algorithms for a model.

Bibliography

[AG98] Ken Arnold and James Gosling. *The Java Programming Language (2Nd Ed.)*. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 1998.

[Eic05] Holger Eichelberger. *Aesthetics and Automatic Layout of UML Class Diagrams*. PhD thesis, Bayerische Julius-Maximilians-Universität Würzburg, 2005.

[EJL+03] Johan Eker, Jörn W. Janneck, Edward A. Lee, Jie Liu, Xiaojun Liu, Jozsef Ludvig, Stephen Neuendorffer, Sonia Sachs, and Yuhong Xiong. Taming heterogeneity—the Ptolemy approach. *Proceedings of the IEEE*, 91(1):127–144, Jan 2003. doi:10.1109/JPROC.2002.805829.

[ES15] Detlef Esslinger and Wolf Schneider. *Die Überschrift : Sachzwänge - Fallstricke - Versuchungen - Rezepte*. Springer VS, fifth edition, 2015.

[FvH10] Hauke Fuhrmann and Reinhard von Hanxleden. Taming graphical modeling. Technical Report 1003, Christian-Albrechts-Universität zu Kiel, Department of Computer Science, May 2010.

[KD10] Lars Kristian Klauske and Christian Dziobek. Improving modeling usability: Automated layout generation for Simulink. In *Proceedings of the MathWorks Automotive Conference (MAC'10)*, 2010.

[KPP08] Dimitrios S. Kolovos, Richard F. Paige, and Fiona A.C. Polack. The epsilon transformation language. In *Theory and Practice of Model Transformations*, volume 5063 of *Lecture Notes in Computer Science*, pages 46–60. Springer Berlin Heidelberg, 2008.

[Lee03] Edward A. Lee. Overview of the Ptolemy project. Technical Memorandum UCB/ERL M03/25, University of California, Berkeley, CA, 94720, USA, July 2003.

[LHB03] William Lidwell, Kristina Holden, and Jill Butler. *Universal principles of design : 100 ways to enhance usability, influence perception, increase appeal, make better design decisions, and teach through design*. Rockport publ. cop., Beverly (Mass.), 2003. URL: http://opac.inria.fr/record=b1128784.

[LN00] Edward A. Lee and Steve Neuendorffer. MoML - a modeling markup language in XML Version 0.4. Technical Memorandum UCB/ERL M00/12, University of California, Berkeley, CA 94720, 2000.

[LS11] Edward A. Lee and Sanjit A. Seshia. *Introduction to Embedded Systems, A Cyber-Physical Systems Approach*. Lulu, 2011. URL: http://LeeSeshia.org.

[LXL01] Xiaojun Liu, Yuhong Xiong, and Edward A. Lee. The ptolemy ii framework for visual languages. In *HCC*, pages 50–. IEEE Computer Society, 2001.

[MELS95] Kazuo Misue, Peter Eades, Wei Lai, and Kozo Sugiyama. Layout adjustment and the mental map. *Journal of Visual Languages & Computing*, 6(2):183–210, June 1995.

[Pal92] Stephen E. Palmer. Common region: A new principle of perceptual grouping. *Cognitive Psychology*, 24(3):436–47, 1992.

[PR94] Stephen E. Palmer and Irvin Rock. Rethinking perceptual organization: the role of uniform connectedness. *Psychonomic Bulletin and Review*, 1(1):29–55, 1994.

[SFvH09] Miro Spönemann, Hauke Fuhrmann, and Reinhard von Hanxleden. Automatic layout of data flow diagrams in KIELER and Ptolemy II. Technical Report 0914, Christian-Albrechts-Universität zu Kiel, Department of Computer Science, July 2009.

[SSM+13] Miro Spönemann, Christoph Daniel Schulze, Christian Motika, Christian Schneider, and Reinhard von Hanxleden. KIELER: building on automatic layout for pragmatics-aware modeling (showpiece). In *Proceedings of the IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC'13)*, San Jose, CA, USA, September 2013.

[SSvH13] Christian Schneider, Miro Spönemann, and Reinhard von Hanxleden. Just model! – Putting automatic synthesis of node-link-diagrams into practice. In *Proceedings of the IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC'13)*, pages 75–82, San Jose, CA, USA, 15–19 September 2013. doi:10.1109/VLHCC.2013.6645246.

[SvH14] Christoph Daniel Schulze and Reinhard 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.

[Wer23] Max Wertheimer. Untersuchungen zur lehre von der gestalt. ii. *Psychologische Forschung: Zeitschrift für Psychologie und ihre Grenzwissenschaften*, 4:301–350, 1923.

[Wil08] Robin Williams. *The Non-designer's Design Book, Third Edition*. Peachpit Press, Berkeley, CA, USA, third edition, 2008.