Project Report for Google Maps for Models

Winter term 2020/21

David Wolff

Master project
March 2021

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

Advised by
M.Sc. Niklas Rentz

Selbstständigkeitserklärung

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

Kiel,

Abstract

The Google Maps for Models project aims to identify and extract the interactive browsing features as commonly used in mapping tools such as Google Maps and apply the concepts to automatically generated diagrams for models as used in the KIELER project to improve the browsing experience of the model visualization. In this context, the focus lies on SCCharts. The main feature is *Smart Zoom*, that automatically expands and collapses regions and complex states based on the zoom level and aborts rendering for invisible elements. Other features are the simplification of small text elements and constant line width independent of the viewport. Using these new concepts, we evaluate the increased rendering performance in medium to large models.

Acknowledgements

I would like to thank Niklas Rentz for giving constructive feedback on both the implementation and the written elaboration as well as helping each week to determine the next steps for the project. I would also like to thank Prof. Dr. Reinhard von Hanxleden for helping to determine possible next features and improvements as well as the general direction of the project. In addition I am thankful for the guidance of the Real-time Embedded Systems group with parts of the KIELER project I was unfamiliar with.

Contents

KEITH	Kiel Environment Integrated in Theia
KIELER	Kiel Integrated Environment for Layout Eclipse RichClient
VSCode	Visual Studio Code
SVG	Scalable Vector Graphics
IDE	Integrated Development Environment
SCChart	Sequentially Constructive Statecharts
IDE	Integrated Development Environments
KLighD	KIELER Lightweight Diagrams
ID	Identifier

Chapter 1

Introduction

The main goal of this project is to apply functionalities seen in Google Maps to the visualization of models in Kiel Environment Integrated in Theia (KEITH) [Dom18; Ren18]. KEITH is a browser-based Integrated Development Environments (IDE) for Model-Driven Development that uses the Kiel Integrated Environment for Layout Eclipse RichClient (KIELER) [HFS11] language server to provide features such as model visualization and simulation for languages such as Sequentially Constructive Statecharts (SCCharts) [HDM+13]. It is developed using the Theia [1] framework and is designed to apply known and well-accepted user experiences in web IDEs such as Visual Studio Code (VSCode). To this end, the generated diagrams should be easy to understand and intuitively to interact with.

In this context, we worked specifically on the visualization of SCCharts. The diagram for the model is represented using the KIELER Lightweight Diagrams (KLighD) [SSH13] framework. The resulting SVG for the diagram and many interactions are realized using Sprotty [2]. In this process, most model updates are handled by the language server. To allow good responsiveness, we will implement all features on the client side and use no new server requests. Additionally, we try to improve the performance as the whole diagram is rendered in its entirety for each frame.

In this work we will look at the motivation for the concepts in Chapter 2, their implementation in Chapter 3 and evaluation in Chapter 4, followed by possible future work in Chapter 5 and a final conclusion in Chapter 6.

[1]https://theia-ide.org/
[2]https://github.com/eclipse/sprotty

Chapter 2

Motivation

When working on increasingly complex models, it is often difficult to comprehend their behaviour and find problems solely based on the textual representation. Therefore model visualizations often allow to gain quick insight of a model. As such they are often a vital component for an efficient workflow as complexity increases.

Diagrams of large models can be barely readable at first glance, when viewed on a computer screen as seen in Figure 2.1. They often contain unreadably small texts, vanishingly small edges and paths as well as information irrelevant to the top level structure. Therefore, an important focus of this project is the simplification and representation of complex states and regions. That allow for a more dynamic view of the structure in question. In addition we will improve visibility for transitions as well as region and state boundaries, as these are important to discern system behaviour. An example of these features can be seen in Figure 2.2.

Moreover, large graphs can lead to low frame rates in KEITH, as the entire diagram is rendered for each translation or scaling step. With the browsing experience in mind, increasing the performance by removing invisible and currently unreadable elements is also an aspect of this project.

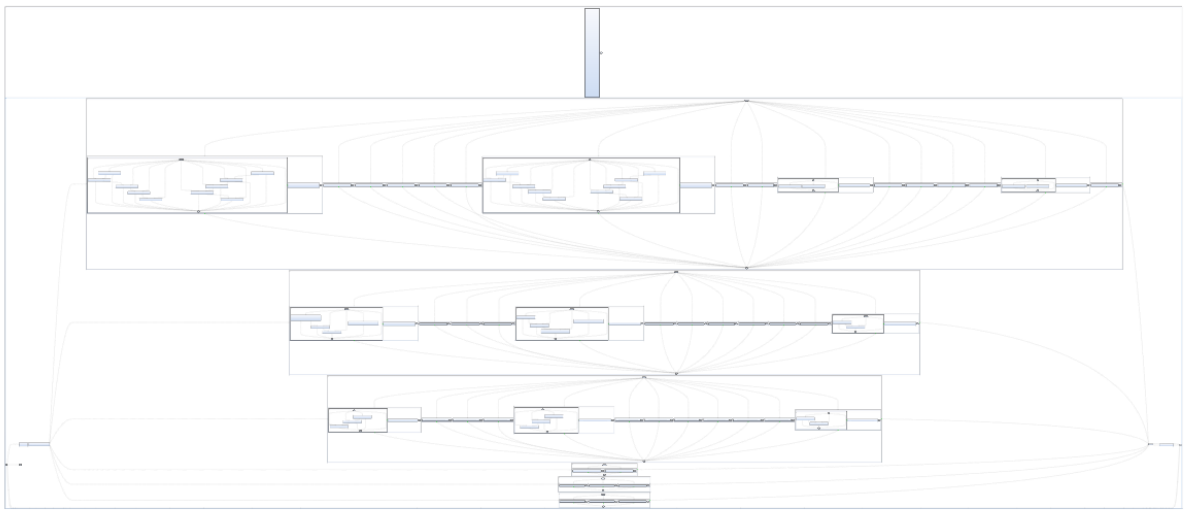


Figure 2.1. Diagram of the wagon model using original KEITH.

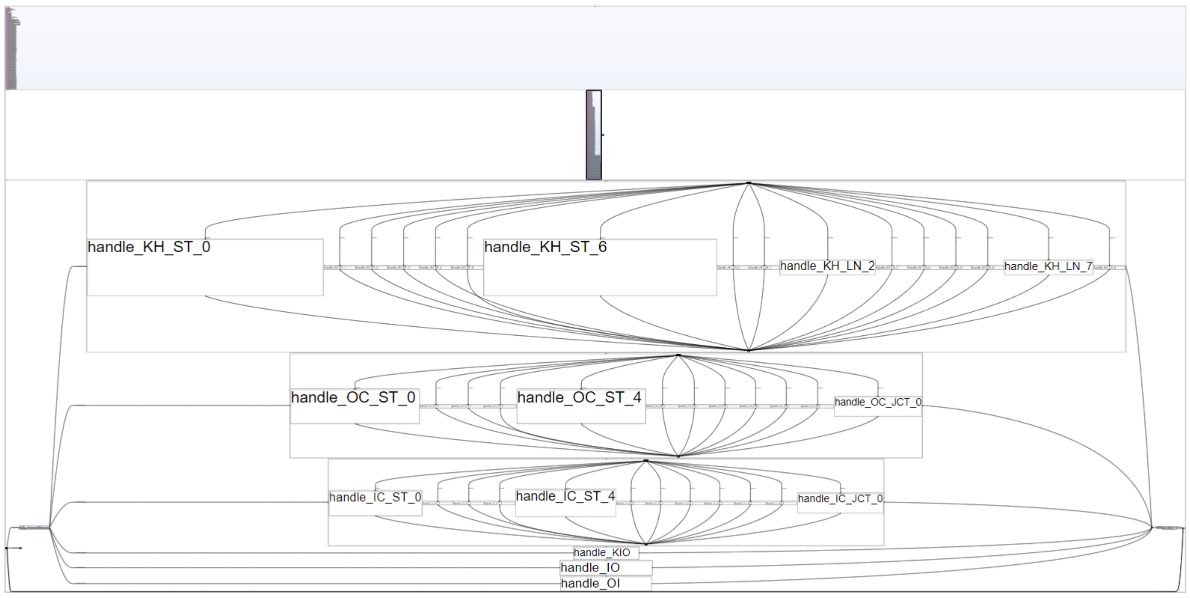


Figure 2.2. Diagram of the wagon model using all new features.

Chapter 3

Implementation

In this chapter, the main aspects of the implementation as well as decisions that were made during the development are presented.

3.1 Scaling of Lines

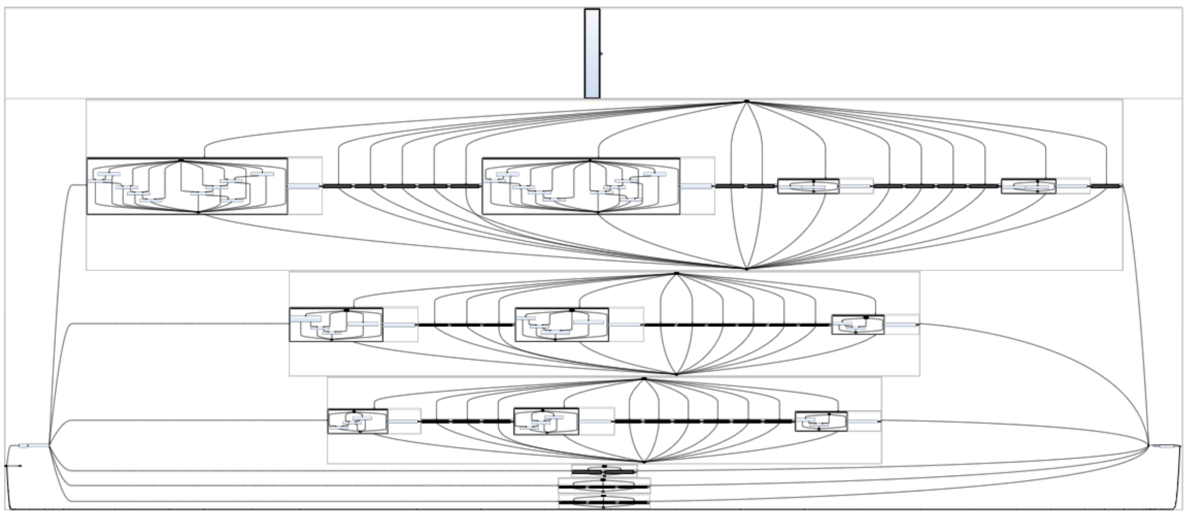
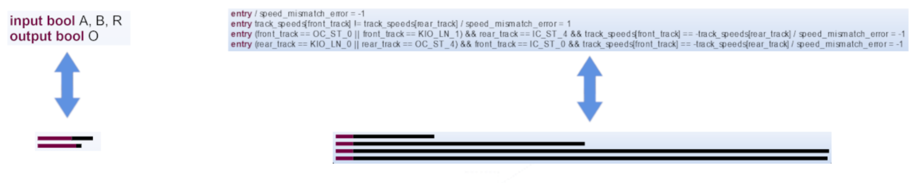


Figure 3.1. Diagram of the wagon model with line scaling.

Much interesting information is reinforced through lines or paths such as the structure of states and transitions. The width of these are predefined and independent of the viewport in KEITH.

Thus, the first change made during the project was to scale the line width depending on the current zoom level as seen in Figure 3.1. Dividing the line width by the current zoom level results in lines being drawn with constant width as a new rendering is made for each scaling step. During this some edge cases need to be handled. The ones considered here are the original expand and collapse buttons as well as drawing lines as originally intended, when zooming in further.

3.2 Simplification of Small Text Elements



```
input bool A, B, R
output bool O
```

```
entry / speed_mismatch_error = -1
entry track_speeds[front_track] != track_speeds[rear_track] / speed_mismatch_error = 1
entry (front_track == OC_ST_0 || front_track == KIO_LN_1) && rear_track == IC_ST_4 && track_speeds[front_track] == -track_speeds[rear_track] / speed_mismatch_error = -1
entry (rear_track == KIO_LN_0 || rear_track == OC_ST_4) && front_track == IC_ST_0 && track_speeds[front_track] == -track_speeds[rear_track] / speed_mismatch_error = -1
```

Figure 3.2. Examples of text simplification.

Text elements take a considerable time to render. As small text elements are unreadable, we simplify the small text elements to allow the user to determine that there is still text present.

We first need to define a threshold under which this should occur. To check the height of a given text element in the view port, we have to divide the server calculated height by the zoom level. To replace the text elements I used simple rectangles with the same width. These are set to fill 50% of the original text height and use the same color style to appear visually close to the original text.

Later I was told that a similar function was already implemented, that hides really small text elements completely. This can already be fine-tuned at a much finer granularity for several different types of text elements and could be expanded to also fill the need for placeholders.

3.3 Collapsing and Expanding

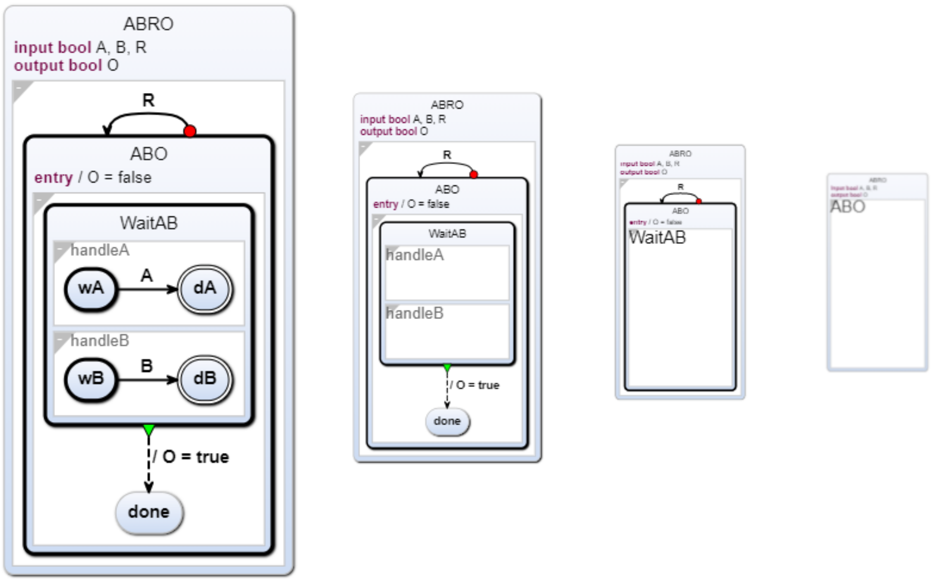


Figure 3.3. Different zoom levels of ABRO diagram using Smart Zoom.

An example of the zoom-dependant expansion state of elements can be seen in Figure 3.3.

To be able to remember and apply expansion of states and regions the Boolean property expansion state is added to `KNode` model elements. This property is checked when rendering a `KNode`. This results in normal behavior, when the expansion state is set to true, indicating an expanded region. Otherwise, the rendering returns no SVG for the current node and consequently does not render nested nodes.

To set the appropriate expansion state, the first test implementation used the `ZoomMouseListener` provided by Sprotty to detect a change in zoom level. After a fixed change in zoom level, the most nested visible nodes got collapsed or expanded based on the direction in the change of the zoom level. To realize this the entire model was traversed and as a result, when changing the expansion state the visualization would stutter.

To mitigate this, a new class `DepthMap` was implemented. Here, node references are combined into regions that should appear and disappear together. To make finding the appropriate elements faster, a 2D array is used to save the regions, where the outer array corresponds to the nesting depth. As such now only each region needs to be checked to apply the expansion states.

For deciding the expansion state based on the size of the region, the bounding rectangle of the region is saved and used to quickly identify a region based on a HashMap with the rectangle Identifier (ID) as the key and the corresponding region as the value. This can be used in the rendering to determine when the next region is being rendered.

In addition, the tree structure of the model is applied to the regions as well by traversing the parents of the regions bounding rectangle until the next parent region is found. These are then saved in the corresponding regions as well. This also allows to minimize the number of regions that need to be checked to apply the correct expansion state as the child regions of collapsed regions do not need to be checked.

The next modification was the process to determine the expansion state of a region. To do this, dependent on the current viewport, the size of the region is rescaled with the zoom level to determine its actual width and height in the rendering. These are then compared to the dimensions of the viewport. The bigger of the two resulting fractions is then used to determine the expansion state by thresholding. This threshold was added to the rendering options and can be changed during browsing. A threshold of 0.5 results in a region being collapsed, when its largest dimension fills under 50% of the viewport and expanded otherwise.

To help with visibility, when using a large threshold, surpassing the native resolution of the diagram through zooming results in all nodes being expanded.

3.4 Scaling of Titles

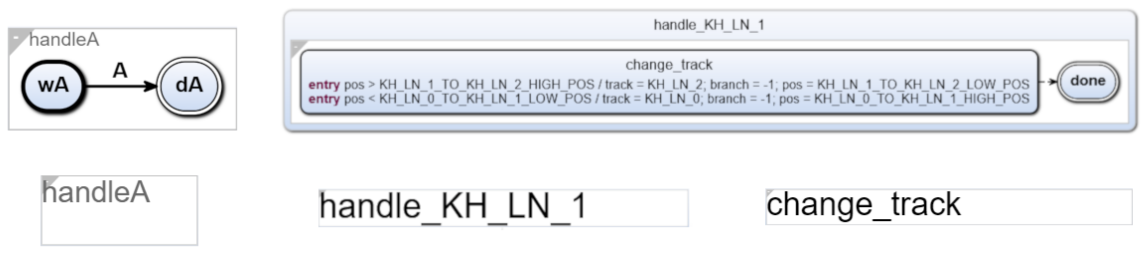

handleA
wA
A
dA
handle_KH_LN_1
change_track
entry pos > KH_LN_1_TO_KH_LN_2_HIGH_POS / track = KH_LN_2; branch = -1; pos = KH_LN_1_TO_KH_LN_2_LOW_POS
entry pos < KH_LN_0_TO_KH_LN_1_LOW_POS / track = KH_LN_0; branch = -1; pos = KH_LN_0_TO_KH_LN_1_HIGH_POS
done
handleA
handle_KH_LN_1
change_track

Figure 3.4. Displaying different title types in order of precedence. Left to right: region title, state title, single macro state title.

For a collapsed region, it is not apparent what role it might have in the model. To have at least an impression of what could be hidden here, adequate titles are searched for and scaled to ensure readability. In this context, three kinds of titles are distinguished. A region itself can have a title as well as a super state. Here a super state has at least some internal states and a macro state has at least some declarations. If none of these apply and the region has only one complex state, this state title is then used as the region title. These three types of titles as well as the final scaling can be seen in Figure 3.4.

The detection of a direct title of a region is possible directly from the model. This is also located in the correct nesting level. This allows to simply scale and indent the `KText`, when a title of a collapsed region is encountered.

For super and macro states, the property `isNodeTitle` was added to `KText` on the server side, which is set to true if it is a macro state title. With this property on the client side it can be checked whether a region has macro states or is a super state.

When the region itself is a super state, the title needs to be rendered in the corresponding child area. For this reason the `KText` is rendered again, one nesting level lower in the rendering. To find the corresponding region in the lower level the ID is used. This rendering is then scaled the same way.

For macro state titles, we have to make sure that there is only one macro state. In order to determine whether the appropriate `KText` can be used as a title. Here the `KText` is rendered one level higher, from the content of the region to the same level as the region. This `KText` then only needs to be scaled.

With this lifting and lowering of the nesting depth of elements, we have to make sure to only perform this step once and employ counter measures to not confuse titles as this could impact whether or not we find multiple macro state titles.

For collapsed regions where no title applies, a placeholder in the form of a magnifying glass is drawn to suggest that more can be displayed here by zooming in.

3.5 Limiting the rendered SVG

The most significant part of the implementation to improve performance for the visualization of large models is not rendering regions. The idea is simple: With the size and position of all regions and the viewport it can be determined which regions are not currently visible. Following this these regions can be collapsed to skip the rendering. However, the implementation turned out to be one of the bigger challenges.

The main problem is that the exact position in the diagram, as well as the viewport is needed. The absolute position of nodes can not be determined easily. For each `KNode`, the KIELER language server pre-calculates position and size. This position is relative to the parent node. Using Sprotty's `getAbsoluteBounds` should be able to determine the absolute position by going through all parent nodes and their relative positions. In contrast Sprotty assumes, that the position of the child node relative to the parent node begins at the corner of the parent node. Thus, the displacements due to titles, declarations, and other layouting information add up with each nesting level. To combat the complexity of going through all layouting information and constructing the absolute position, a simpler approach was implemented.

It is not technically pretty, but serves its purpose. A function from Sprotty called `getAbsoluteClientBounds` is used for inspiration here. It looks up the SVG associated with a graph element and determines the position relative to the entire browser window. With this being done for each region, it introduces a lot of new operations for the initial rendering and thus is a bottleneck for diagram generation of larger models. This could be circumvented, by using the already available layouting information as elaborated in Chapter 5.

The indexing structure assumed by Sprotty differs from the one implemented by KIELER. Thus for each region, the SVG of the bounding rectangle, as well as the diagram window must first be found. To calculate the position of the region relative to the diagram.

The first approach was to try to determine the position of all regions, before the centering and scaling action of the initial drawing. This caused two problems. First, after generating the first diagram, micro layouting information are still exchanged with the server. Second, the centering and scaling animation of Sprotty, executed after the first complete rendering, cannot be paused easily for just this purpose. In addition the procedure of centering, after the initial drawing might change in the future.

To circumvent the first problem, we find the step where no more micro layouting information is exchanged. To do this the data structure used to store this information is checked to see if it is empty. Then the finished diagram should be drawn at the client.

The second problem with the animation is compensated through scaling to original proportions. To do this the previously acquired browser position is put into reference to the diagram window. Then the current dimensions and position of the element are rescaled using the zoom level.

However, there are sometimes position calculations for some elements in some renderings with a large error. To still get a accurate position, an error margin is introduced and elements that exceed this margin are checked again in subsequent renderings, until a good position approximation is found. Therefore, it is important to get an error margin that is large enough to compensate the error and allow as few absolute position approximations as possible. With the error margin also extending the range of the visibility for drawn elements, it still needs to be small enough to improve performance. These extra steps would not be needed, if the absolute position was calculated using the layouting information as explained in Chapter 5.

Chapter 4

Evaluation

The goal of this project is to achieve a browsing experience similar to mapping software such as Google Maps and apply them to models. To this end, we introduced a dynamic way for states and regions to be hidden based on their size and position in the current rendering, while still retaining some information through their titles. Furthermore, the visibility of transitions and boundaries are enhanced as well. This makes the top level view more accessible and unveils previously unrecognizable information. Furthermore this persists, when viewing larger parts of a model.

Contrary to this is the unpolished look of the new features in comparison to the visual style of KIELER. In addition the expansion and collapse actions are not novel functionality, but are realized in a client-side way with the aim to decrease performance overhead through these actions. With this, the project could still be used for further work.

4.1 Performance

To allow for a smooth browsing experience responsiveness is essential. Therefore, we will take a look at the performance change due to the new features.

Figure 4.1 visualizes the frametimes when browsing through the railway wagon model [1] with and without the new features. Here we can see an improvement of almost a third just by using the simplified text elements. The combination of collapsing and expanding regions as well as hiding currently invisible elements can result in a refresh rate almost three times as fast as the original implementation. With both combined, we get a relatively smooth refresh rate for all zoom levels especially, when viewing the entire diagram. However, the `DepthMap` and the process for finding the absolute position create a considerable overhead when trying to visualize larger models. This not only leads to longer initialization times, but also can reach a tipping point, where the collapse and hiding of elements can worsen the performance. Therefore these results should be viewed with caution and future work can ensure a performance increase for all diagrams.

[1] This model is from the railway project of summer semester 2017.

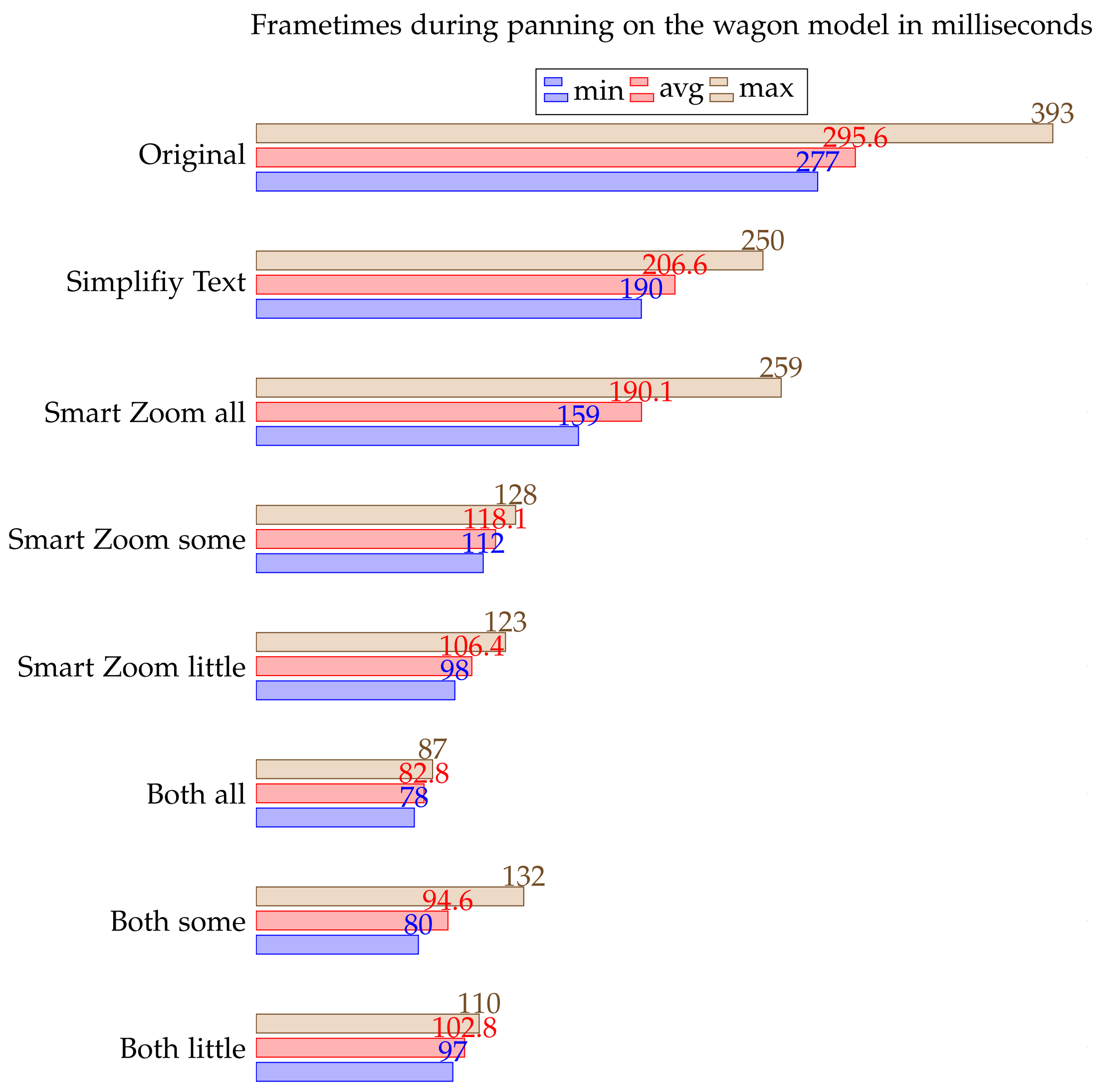


Figure 4.1. Frametimes were measured using Google Chrome performance recording and rounded to whole numbers. For each test 10 consecutive frametimes were randomly taken during continuous scrolling.

Chapter 5

Future Work

The results can be improved further in future work on this topic. In addition, other topics for future work were encountered.

Considering the visualization, regions pop in and out of the diagram abruptly. This could be improved by morphing or smooth changes in visibility. In addition, the titles are just scaled versions of `KText` elements without further modification or a smooth transition into their normal occurrences. For example, it would be nice to have a type of visual difference between the three types of titles for clarity. To go even further, the concept of placeholders could be expanded for several different types of structures. For example placeholders indicating no further nesting or conveying the complexity of the hidden states and regions. To do this, the transitions could also be adjusted to fit to the new placeholders.

Actions such as panning and zooming get pipelined by Sprotty. This can lead to the build up of actions, that get executed sequentially. This results in unresponsiveness of visualizations with slow update rates. To reduce this effect, the `ActionDispatcher` of Sprotty would need to be modified to combine confluent actions. A simple example would be to combine subsequent translations into one by adding their displacement values and replacing them.

Another possible improvement considering Sprotty is to only generate a new SVG, when a considerable change in the viewport was made or an event was triggered. This would allow almost instantaneous updating, when just scrolling through a diagram.

The calculation of absolute positions can be implemented by evaluating all layouting information. The main problem here is to find all relevant layouting information for each element and transform them into absolute displacements with respect to the parent element. Then you could determine the absolute position of a child element by going through all parents and adding their displacements.

The overhead of the `DepthMap` could be reduced by progressively building it top down, i. e. that the visualization begins at the root and elements are only added, when the corresponding region or state gets expanded. Another way would be to limit it to just the visible elements and use the tree structure to find regions that could get visible by going through the relevant parents, when scrolling or zooming occurred.

The model could be generated progressively with server requests, with elements only transmitted, when needed as well.

Lastly, the layout of regions and super states could be modified to better fit the current viewport and allow for a more interactive experience. An idea for this is to find appropriate positions and sizes for regions and super states and then scale the child area contents of these to fit into the child regions. If the style of the diagram elements, could be adjusted to this change in size, the viewing experience should still be smooth, while there is more freedom to try to optimize the overall layout further.

Chapter 6

Conclusion

With only one student participating in the project, the scope of possible implementations was limited. To this end, the implemented features could be viewed as a proof of concept, that still need some work. Those are the dynamic client based collapse and expansion of states and regions, reducing the rendering scope to only visible elements, finding and processing title candidates as well as text placeholders and constant line widths. These improve the comprehensibility on the top level view and the performance of the visualization for some models, but struggle when the models get too complex. In addition to the explored features many new possible topics were discovered or rediscovered.

Bibliography

[Dom18] Sören Domrös. "Moving model driven engineering from eclipse to web technologies". In: (Nov. 2018), p. 102. URL: https://rtsys.informatik.uni-kiel.de/~biblio/downloads/theses/sdo-mt.pdf (visited on 03/25/2021).

[HDM+13] Reinhard von Hanxleden, Björn Duderstadt, Christian Motika, Steven Smyth, Michael Mendler, Joaquín Aguado, Stephen Mercer, and Owen O'Brien. *SCCharts: Sequentially Constructive Statecharts for safety-critical applications*. Technical Report 1311. ISSN 2192-6247. Christian-Albrechts-Universität zu Kiel, Department of Computer Science, Dec. 2013.

[HFS11] Reinhard von Hanxleden, Hauke Fuhrmann, and Miro Spönemann. "KIELER—The KIEL Integrated Environment for Layout Eclipse Rich Client". In: *Proceedings of the Design, Automation and Test in Europe University Booth (DATE '11)*. Grenoble, France, Mar. 2011.

[Ren18] Niklas Rentz. "Moving transient views from eclipse to web technologies". In: (Nov. 2018), p. 103. URL: https://rtsys.informatik.uni-kiel.de/~biblio/downloads/theses/sdo-mt.pdf (visited on 03/25/2021).

[SSH13] C. Schneider, M. Spönemann, and R. von Hanxleden. "Just model! — putting automatic synthesis of node-link-diagrams into practice". In: *2013 IEEE Symposium on Visual Languages and Human Centric Computing*. 2013 IEEE Symposium on Visual Languages and Human Centric Computing. ISSN: 1943-6106. Sept. 2013, pp. 75–82. DOI: 10.1109/VLHCC.2013.6645246.