Diagram Control and Model Order for Sugiyama Layouts

Sören Domrös[0000−0002−8011−8484] and
Reinhard von Hanxleden[0000−0001−5691−1215]

Department of Computer Science, Kiel University, Kiel, Germany
{sdo,rvh}@informatik.uni-kiel.de

Abstract. Graphical WYSIWYG editors for programming languages are popular since they allow to *control* the diagram layout to express *intention* via *secondary notation* such as proximity and topology. However, such editors typically require users to do manual layout. Conversely, automatic layout of diagrams typically fails to capture intention because graphs are usually considered to not contain any order. *Model order* can combine the desire for control of secondary notation with automatic layout, without additional overhead, since the textual model already employs secondary notation. We illustrate how model order can exert control on the example of the programming languages SCCharts and Lingua Franca and collect developer feedback to validate our findings.

Keywords: Automatic Layout · Model Order · User Intentions.

1 Introduction

Automatic layout rises in popularity, as seen in the example of elkjs[1] with, as of this writing, more than 700.000 weekly downloads. Even though automatic layout improved over time and gets more widely used, WYSIWYG editors that rarely employ automatic layout are still very common. WYSIWYG editors place the burden of layout on the user, which can be a severe impediment to productivity [10]. However, WYSIWYG editors have the advantage that they allow controlling secondary notation [10] by creating order, grouping, or alignment in a very direct way and on a graphical level, which is desirable. As Taylor reported for WYSIWYG type-setting [16]: "People like having feedback and control."

One approach to augment automatic layout of diagrams with control is to let the user formulate explicit layout constraints, e. g., through (textual) model annotations or via some WYSIWYG-like graphical interaction [5,11]. Constraints have the advantage to be integrated into layout algorithms. Hence, layout does not need to be done on a pixel granularity, which when done manually often has inconsistencies [14]. Instead, they focus on topology, alignment, or proximity of nodes. Constraints, however, require additional effort beyond creating a textual

[1] https://npmtrends.com/elkjs

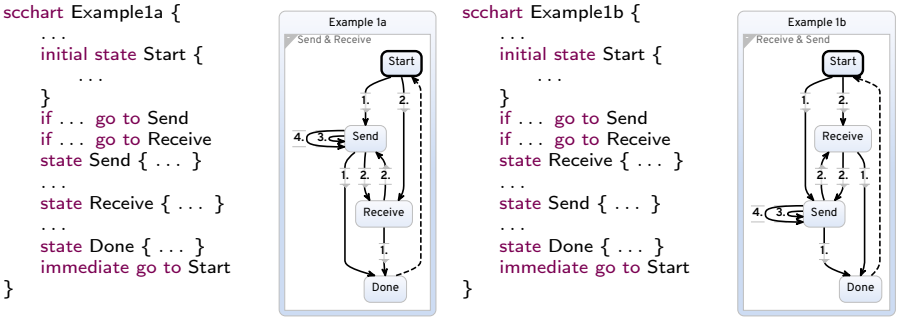


Figure 1: Two semantically identical variants of an obfuscated SCCharts model created by and used with permission of Scheidt & Bachmann System Technik GmbH.

model, and sometimes require knowledge about the underlying layout algorithm to be used effectively, as reported by users and developers of ELK [4].

In this paper, we investigate the research question how to exert control by using *model order* [2] for *Sugiyama or layered layouts* [15] whenever the textual model in a tool that employs modeling pragmatics [7] by using a textual model and a graphical model side-by-side expresses secondary notation (R1). We also explore how model order integrates with the mental map, layout stability, aesthetic criteria, and secondary notation in text and diagram (R2).

The Sugiyama algorithm consisting of the phases cycle breaking, layer assignment, crossing minimization, node placement, and edge routing may produce drawings as the one depicted in Fig. 1. Here, nodes are assigned to (horizontal) layers such that nodes in the same layer are not connected by an edge. The cycle breaking step determines which edges should go against the layout direction and hence the order of Send and Receive. Blindly optimizing aesthetic criteria disregards the intention of the model and leaves no leverage to control the layout. The two versions are semantically identical, however, the vertical ordering in Fig. 1b suggests that Send happens before Receive while Fig. 1d suggests the inverse. Fig. 1b was created using the textual model in Fig. 1a, while Fig. 1d was created using Fig. 1c. Hence, the textual order should be considered, since the mental map, the inner representation of the model, and secondary notation of the textual model should not diverge from their representation in the diagram.

To answer the research question R1 and R2 stated above, we contribute

- an investigation how controlling the diagram via model order influences the mental map, layout stability, and secondary notation in Sec. 3 and
- an analysis of SCCharts and Lingua Franca (LF) in the context of model order and control in Sec. 3.1 and 3.2.

Sec. 4 summarizes and generalizes ours insights and suggests future research.

A long version of this paper [3] contains additional examples and insights for model order and its interaction with the mental map, secondary notation, stability, aesthetic criteria, and control, more detailed evaluation results, and a guidebook how to get model order information for a given language.

2 Related Work

Purchase [13] investigated what aesthetic criteria humans adhere to when drawing graphs given by a textual description. As stated by Purchase, other studies [12,8] focus on reduction of edge crossings, symmetry, placement of important nodes at the top, large angles between incident edges, and average edge length. Purchase investigated whether there are additional criteria people favor when drawing graphs, by analyzing the final drawings and the intermediate steps of two graphs created by the participants. The study revealed that people prefer to place nodes on a grid and that they initially use aesthetic criteria such as only vertical or horizontal edges or nodes ordered lexicographically or by their occurrence in the graph representation. Moreover, the study revealed that the participants worked through the graph node by node or edge by edge depending on the graph representation. They intuitively used the given model order of the graph representation and revised this order partially if the result created undesired crossings or clutter. To evaluate model order, we hence need real models and preferably the developers that built them to investigate what their intention in specific placements was. Moreover, evaluation of model order configurations should be done in an interactive tool to considering the different creation steps.

3 Control and Model Order

How model roder relates to common aesthetic criteria was already covered by previous work [1,2]. Here, we investigate how model order relates to mental map, stability, secondary notation, and control on a meta level (R2) by focusing on tools for model-driven-engineering that use text and diagram side-by-side utilizing the concept of modeling pragmatics [7].

The *mental map* describes the mental image one has of a graphical model [9]. Misue et al. define preserving the mental map as preserving orthogonal ordering and proximity. Preserving the mental map and with it the *stability* of the drawing is especially important when working with models for real use-cases since they are typically large, hierarchical, and do not fit on a single screen[2]. Developers already create a mental map while creating the textual model and not only by looking at the accompanying diagram. Hence, we might compromise the mental map if the textual model does not match the graphical model, as it would be the case if Fig. 1a would create the diagram in Fig. 1d. Since the textual ordering

[2] This can be explored interactively in the LF playground https://github.com/lf-lang/playground-lingua-franca by opening the cdn_cache_demo model created by Magnition taken from https://github.com/MagnitionIO/LF_Collaboration.

only has one dimension but the drawing has two to express order, we have to further determine which dimension in the diagram corresponds which textual ordering for a given language.

Similarly, *secondary notation* exists in textual models. E. g., developers begin to write the textual model with an initial state at the top, final states are typically at the bottom, and nodes that should be next to each other in the drawing are typically also placed next to each other in the textual model, as seen in Fig. 1. Since intentional secondary notation exists in the textual source, we should control the layout using the textual model order to bring secondary notation from the text into the diagram.

Control is a very desirable aspect of layout. This might be the reason WYSIWYG editors are still implemented even though moving boxes around is a tedious process [10]. Moving boxes to desired positions directly creates secondary notation. This level of control can also be achieved using interactive constraint frameworks [5,11] with the advantage of automatic layout. However, using model order inside a layout algorithm can similarly exert control by treating the textual ordering as a constraint. Moreover, model order can also be used as a tie-breaker together with common aesthetic criteria creating different levels of control [2].

To understand how a language can be controlled using model order, we have to identify what part of the textual model is intended secondary notation and how this should be transferred to the two-dimensional diagram during the cycle breaking and crossing minimization steps of the Sugiyama algorithm.

3.1 Controlling SCCharts Layout via Model Order

SCCharts [6] is a sequentially constructive statechart dialect that models control-flow. Fig. 1 depicts an SCChart drawing with the corresponding textual model. An SCChart consists of a declaration of inputs, outputs, constants, variables and actions, which are here filtered out of the diagram. Moreover, it has concurrent regions (the white box called Send & Receive in Fig. 1b) with states and transitions between them. The textual syntax only allows to define outgoing transitions of a state directly under the state declaration, which constraints the global edge order. E. g., the state Start has two outgoing transitions defined below it. States might have internal behavior including everything an SCChart may consist of.

The developer may reorder states without changing the semantics of the model. However, the initial state, i. e., the state Start, is usually defined at the top of a textual model. Moreover, the textual SCCharts model employs secondary notation such that the node model order indicates the control-flow direction. E. g., the textual model for Fig. 2 defines the states Ok and Active that belong to the same control-flow branch above the state Failed, Disconnected defined first, and the states Inactive and Disconnect at the end.

Only transitions with mutually exclusive transition guards can be freely reordered without changing the behavior since reordering changes their priority, which SCCharts users want to visualize as secondary notation (R2). Hence, if

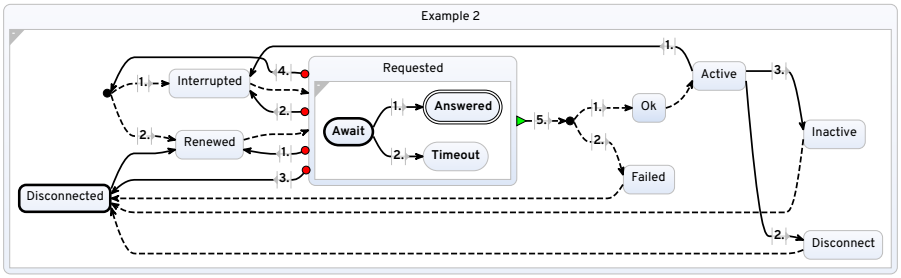


Figure 2: An obfuscated SCCharts model created by and used with permission of Scheidt & Bachmann System Technik GmbH. It uses model order to constrain the flow and as tie-breaker for the (vertical) order inside a layer.

the textual ordering of model elements has semantics, they cannot be freely reordered to exert control (R1).

Previous work on model order for SCCharts [1,2] resulted in the following model order configurations. Strategy 1 uses the strict model order cycle breaker that enforces the node model order along edges since the node model order employs intended secondary notation. Strategy 1 uses the edge model order to pre-order node and edges before the crossing minimization step, which may revise the pre-order to reduce edge crossings. Additionally, Strategy 1 uses the node and edge model order as secondary criterion for crossing minimization such that the solution with minimal crossings that respects most of the model order will be chosen. Strategy 1 was already field-tested while teaching two students courses about Embedded System and Synchronous Languages and is currently also employed by Scheidt & Bachmann System Technik GmbH (S&B). Strategy 2 has a different use-case and fully controls the layout by model order to create layouts for documentation [2].

Additionally to our own experience with SCCharts, we interviewed five S&B developers about 36 SCCharts models created for non-safety critical projects in the railway domain. 30 models fully conformed to the model order not counting models where one edge needed reordering to achieve the desired result.

On no occasion, developers or students reported anything out of the ordinary or problems as a result of the cycle breaking step constrained by model order, e. g., backward edges that should not be there. However, S&B developers did not notice that the node model order constrained the flow of the diagram before pointing it out via Fig. 2. Here, the connector state (black dot) above the initial state Disconnected is a dangling node—a node with only edges to the right—even though it is not a source node. Moving this node somewhere else may, however, hide the symmetry of the Interrupted and Renewed states.

The strict model order cycle breaking has been active as the default layout for about a year, which allowed us to gather feedback on it. During this time this strategy did not produce Obviously Non-Optimal (ONO) or left developers confused. Hence, model order should per default control the flow (R1).

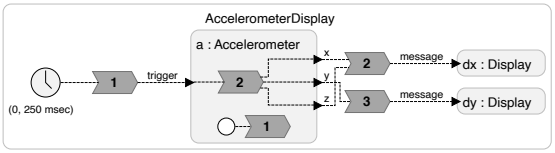
Strategy 2 cannot be the default option for SCCharts, since it may produce ONO layouts. However, if a controlling, more strict, strategy controls most parts of the layout, a bad layout often points to a bad textual model. How this may affect developers needs further evaluation.

3.2 Controlling Lingua Franca Layout via Model Order

Lingua Franca [7] is a polyglot coordination language for reactive real-time systems that models data-flow. An example LF model can be seen in Fig. 3.

```
main reactor {
    a = new Accelerometer();
    dx = new Display(row = 0);
    dy = new Display(row = 1);
    timer t(0, 250 msec);
    state count:int(0);
    reaction(t) →a.trigger . . .
}
```

(a)



(b)

Figure 3: The AccelerometerDisplay LF example model with abbreviated textual model and graphical representation.

Fig. 3a depicts the main reactor of the LF model and Fig. 3b depicts the corresponding diagram with a timer (clock), reactions (numbered arrows shapes), and reactors (gray boxes). Users may define states, actions, timers, reactors, edges between reactors, and reactions and often define the components in the mentioned order forming *ordering groups*. E. g., users define reactors in a separate group above the ordering group of reactions. A reactor may again consist of everything the main reactor has to offer but may define inputs and outputs. Edges between reactors and other elements are created implicitly based on their interfaces, i. e., their declared inputs and outputs, which corresponds to the port model order. The reactor instantiation order can be freely changed as it is the case for all elements despite reactions and typically expresses desired secondary notation that could be controlled via model order (R1). The order of reactions defines their scheduling order. Hence, we cannot freely reorder them to control the layout (R1). However, this scheduling order is secondary notation that developers want to represent in their drawing.

Interviews and presentations as part of the weekly LF team meeting, interviews with Magnition that want to use LF to model cache structures, and one-on-one interviews resulted in the following statements.

1. Edge-crossings should be reduced.
2. Stability is important for large hierarchical models.
3. The ports of reactors define their interface and should hence be respected.
4. The reactor-instantiation-order could be used to control their ordering.
5. Actions should be placed such that the flow indicates their activation.

Note that statement 1, 2, and 3 came from Magnition developers and 1, 3, 4 from the person in the LF community that mainly interacted with us during the presentations and discussions. All the statements above were additionally verified by one-on-one interviews with a small group of developers regarding the layout of LF models.

We see that LF developers value common aesthetic criteria (statement 1) but prefer stability create by model order for large models (statement 2). Reactor order and port order are the primary elements that can be used to control the layout (statement 3 and 4). However, different model elements are not directly comparable, even though there might be relative orderings between the different ordering group (statement 5).

We conclude that we should not use model order strategies that completely constrain the layout for LF (R1). Layout algorithms for LF should use model order only as a tie-breaker as the default strategy and may only constrain nodes of the same ordering group to avoid misleading secondary notation. For cycle breaking the depth-first strategy that uses model order to determine the visiting order proved to be good to handle the often intertwined action and reaction networks quite well. Since most models are small, finding and analyzing all cycle might be a promising strategy. Additionally, big models might want to constrain the port or node order by model order to increase stability and preserve the mental map (R2).

4 Conclusion

How SCCharts and LF layout and secondary notation can be controlled by model order can be generalized for similar languages, as SCCharts are just state-machines and LF is only a special kind of actor oriented data-flow language. Hence, constraining the flow of all state-machines by the state model order creates intentional backward edges for all state-machine dialects. However, the model order should only be used to control the layout and used as a constraint if the underlying textual ordering matches the user intention and can be controlled by the user. Model elements that are ordered by convention, restrictions in the grammar, or are only programmatically created should only control the diagram if their order expresses intention.

While flow of information can often be constrained to create secondary notation that visualizes how data or control may go backwards, constraining crossing minimization by model order should not be a default strategy for automatic layout. Even if model order is only considered as a tie-breaker it always increases stability, which helps to maintain the mental map, and may be one option to control the layout. The control given by strict model order strategies is, however, always desired to create specific layouts for documentation or presentations.

For languages with cross-hierarchical edges and expandable and collapsible hierarchical nodes, stability is a very important aspect. Considering node or port order to constrain the layout can solve this problem without an editing overhead.

References

1. Domrös, S., von Hanxleden, R.: Preserving order during crossing minimization in Sugiyama layouts. In: Proceedings of VISIGRAPP 2022 - Volume 3: IVAPP. pp. 156–163. INSTICC, SciTePress (2022). https://doi.org/10.5220/0010833800003124
2. Domrös., S., Riepe., M., von Hanxleden., R.: Model order in Sugiyama layouts. In: Proceedings of VISIGRAPP 2023 - Volume 3: IVAPP. pp. 77–88. INSTICC, SciTePress (2023). https://doi.org/10.5220/0011656700003417
3. Domrös, S., von Hanxleden, R.: Diagram control and model order for sugiyama layouts (2024). https://doi.org/10.48550/arXiv.2406.11393
4. Domrös, S., von Hanxleden, R., Spönemann, M., Rüegg, U., Schulze, C.D.: The Eclipse Layout Kernel (2023). https://doi.org/10.48550/arXiv.2311.00533
5. Dwyer, T., Marriott, K., Wybrow, M.: Dunnart: A constraint-based network diagram authoring tool. In: Revised Papers of GD '08. LNCS, vol. 5417, pp. 420–431. Springer (2009). https://doi.org/10.1007/978-3-642-00219-9
6. von Hanxleden, R., Duderstadt, B., Motika, C., Smyth, S., Mendler, M., Aguado, J., Mercer, S., O'Brien, O.: SCCharts: Sequentially Constructive Statecharts for safety-critical applications. In: Proc. ACM SIGPLAN PLDI '14. pp. 372–383. ACM, Edinburgh, UK (Jun 2014). https://doi.org/10.1145/2594291.2594310
7. von Hanxleden, R., Lee, E.A., Fuhrmann, H., Schulz-Rosengarten, A., Domrös, S., Lohstroh, M., Bateni, S., Menard, C.: Pragmatics twelve years later: A report on Lingua Franca. In: Proceedings of ISoLA' 22. LNCS, vol. 13702, pp. 60–89. Springer, Rhodes, Greece (Oct 2022). https://doi.org/10.1007/978-3-031-19756-7_5
8. Huang, W., Eades, P., Hong, S.H., Lin, C.C.: Improving multiple aesthetics produces better graph drawings. JVLC **24**(4), 262–272 (2013). https://doi.org/10.1016/j.jvlc.2011.12.002
9. Misue, K., Eades, P., Lai, W., Sugiyama, K.: Layout adjustment and the mental map. JVLC **6**(2), 183–210 (Jun 1995). https://doi.org/10.1006/jvlc.1995.1010
10. Petre, M.: Why looking isn't always seeing: Readership skills and graphical programming. Communications of the ACM **38**(6), 33–44 (Jun 1995). https://doi.org/10.1145/203241.203251
11. Petzold, J., Domrös, S., Schönberner, C., von Hanxleden, R.: An interactive graph layout constraint framework. In: Proceedings of VISIGRAPP 2023 - Volume 3: IVAPP. pp. 240–247. INSTICC, SciTePress (2023). https://doi.org/10.5220/0011803000003417
12. Purchase, H.C.: Which aesthetic has the greatest effect on human understanding? In: Proceedings of GD '97. LNCS, vol. 1353, pp. 248–261. Springer (1997)
13. Purchase, H.C.: A healthy critical attitude: Revisiting the results of a graph drawing study. Journal of Graph Algorithms and Applications **18**(2), 281–311 (2014). https://doi.org/10.7155/jgaa.00323
14. Purchase, H.C., Archambault, D., Kobourov, S., Nöllenburg, M., Pupyrev, S., Wu, H.Y.: The turing test for graph drawing algorithms. In: Proceedings of GD '20. pp. 466–481. Springer (2020). https://doi.org/10.1007/978-3-030-68766-3_36
15. Sugiyama, K., Tagawa, S., Toda, M.: Methods for visual understanding of hierarchical system structures. IEEE Transactions on Systems, Man and Cybernetics **11**(2), 109–125 (Feb 1981). https://doi.org/10.1109/TSMC.1981.4308636
16. Taylor, C.: What has WYSIWYG done to us? The Seybold Report on Publishing Systems **26**(2) (Sep 1996)