Sterling: A Web-Based Visualizer for Relational Modeling Languages

Tristan Dyer[1(✉)] and John Baugh[2]

[1] Brown University, Providence, RI, USA
tristan_dyer@brown.edu
[2] North Carolina State University, Raleigh, NC, USA

Abstract. We introduce Sterling, a web-based visualization tool that provides interactive views of relational models and allows users to create custom visualizations using modern JavaScript libraries like D3 and Cytoscape. We outline its design goals and architecture, and describe custom visualizations developed with Sterling that enable verification studies of scientific software used in production. While development is driven primarily by the Alloy community, other relational modeling languages are accommodated by Sterling's data agnostic architecture.

Keywords: Alloy · Sterling · Formal methods · Visualization

1 Introduction

Model finding tools like Alloy enable a lightweight approach to design and reasoning about complex software systems. Such tools provide push-button analysis for both checking assertions within bounded scopes, and for generating instances that satisfy a property of interest. An attractive feature of Alloy is the immediate feedback provided by visualizations, allowing users to inspect instances and counterexamples in order to identify design problems. The ability to communicate visual information *intuitively* therefore plays a key role in determining the effectiveness of interactions with the user [5].

The built-in visualizer in the Alloy Analyzer can display an instance as a directed graph in which nodes represent atoms and edges represent tuples of relations. To help users better understand an instance, basic properties of the graph such as color and shape can be customized, and graph nodes can be repositioned manually to achieve a clearer layout. Additionally, the graph view supports "projection," a feature most commonly used with models of dynamic systems, in which an instance is displayed from the perspective of an atom or set of atoms. When an instance of such a model is projected over time, the user is able to step through snapshots of individual states in sequence.

Despite these capabilities, some instances can be difficult to interpret as models grow in size and complexity. Some well-known issues, for instance, include the inability to drag nodes out of the rows into which they are initially laid out [3,8].

A. Raschke and D. Méry (Eds.): ABZ 2021, LNCS 12709, pp. 99–104, 2021.
https://doi.org/10.1007/978-3-030-77543-8_7

In addition, the graph layout is recalculated any time a new instance is generated or the projection is changed, so the user is forced to reinterpret the entire graph if, for example, they are stepping through the state atoms in sequence [3,9,12]. Various approaches have been proposed to address these and other issues, either by extending the existing visualizer [12] or by introducing new tools [3,5,8]. Our own experiences with Alloy in the field of scientific computing have highlighted the need for better visualization approaches in general, and for an interface that can also depict *spatial* relationships—not just topological ones—while maintaining *consistency* in those relationships when dynamic updates occur, as they do in problems with time-varying state.

For instance, in one such study, Baugh and Altuntas [2] use Alloy to explore implementation choices and ensure soundness of an extension made to a large-scale storm surge application used in production. To be physically meaningful, models representing finite element meshes—which can be thought of as a triangulation of a surface—are constrained to include only those that have a planar embedding, and therefore do not contain overlapping triangles. Working with relational depictions alone means "untangling" each instance as it occurs, leading to the study's conclusion that "more than any extension to Alloy, what would have benefited our study most is a tool capable of automatically producing planar embeddings of meshes from Alloy instances, which proved to be tedious to do by hand."

A subsequent study [4] with Alloy focused on bounded verification of sparse matrix formats, which use array indirection and other structure to avoid storing zeroes. Dense matrices are modeled as relations mapping indices to values, producing dozens of tuples that clutter and overrun any visualization attempt with edges. The sparse matrices themselves, and the dynamic state changes that accompany them for operations like matrix multiplication, make visualizations nearly impossible to interpret.

2 Sterling Design and Architecture

Motivated by these studies, and drawing on feedback and suggestions from the 2018 Workshop on the Future of Alloy, we have developed an approach that builds on the strengths of existing visualizations. Sterling's design is based on the following principles: the visualizer should (1) implement and extend the capabilities already present in Alloy, (2) employ a modern architecture built using popular languages and well established libraries, and (3) provide functionality for creating domain specific visualizations.

Consistent with these principles, Sterling is a web application,[1] built upon a popular web technology stack using the React and Redux libraries, and packaged with a custom build of Alloy. A client-server relationship is established between Sterling and Alloy by an embedded web server, enabling instances to be immediately visualized in Sterling as they are generated by Alloy. The user interface is similar to Alloy's own, providing graph and table views which extend the

[1] A Sterling demo with examples can be found at https://sterling-js.github.io.

functionality of their counterparts in Alloy, while adding a "script" view that provides users with the ability to create custom visualizations from instance data by writing JavaScript code. Communication between Alloy and the individual views is managed using a mediated model-view architecture, illustrated in Fig. 1. Consequently, other relational logic and model finding tools may also employ Sterling for visualization, so long as data is provided to Sterling in the Alloy XML format.

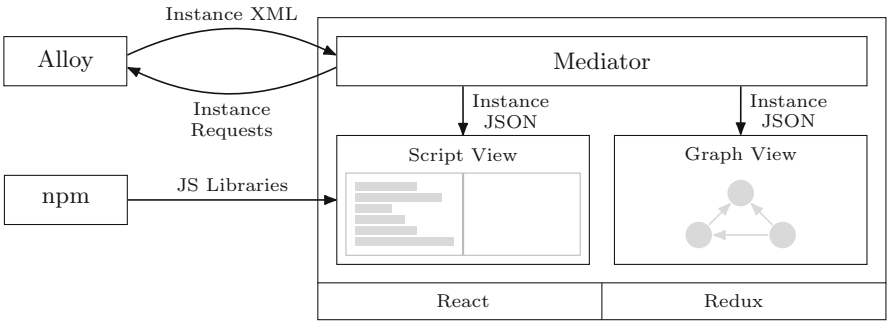


Fig. 1. The sterling architecture.

The Sterling graph view offers the same functionality as the Alloy graph view, but also provides a few key extensions. Most notably, graph elements are not restricted to rows, and users may freely arrange graphical elements to make the display more readable. Furthermore, the layout algorithm is not automatically executed when the projection is changed or a new instance is generated, and so graphical elements remain static as users step through stateful models and generate instances.

The Sterling script view provides an environment for the rapid development of custom visualizations by bringing together a text editor, a blank canvas, and a JavaScript execution environment, giving users a basic "code sandbox" in which they can create visualizations based on instance data using their favorite JS libraries. Within the script view environment, all instance data—the signatures, fields, atoms, and tuples—are exposed as JS variables. Additionally, users have direct access to the npm package repository, which can be used to add visualization (or any other useful) libraries to the scripting environment. This combination enables, for example, a user to bind atoms to shapes using the D3 visualization library, and to calculate their positions based on relationships defined by tuples. We have found this paradigm to be particularly useful for visualizing instances of models with inherent spatial properties. For example, a planar embedding of a finite element mesh, as previously described, is shown in Fig. 2. More custom visualizations, including ones for sparse matrices and some common puzzles, can be found in the interactive demo on the Sterling website.

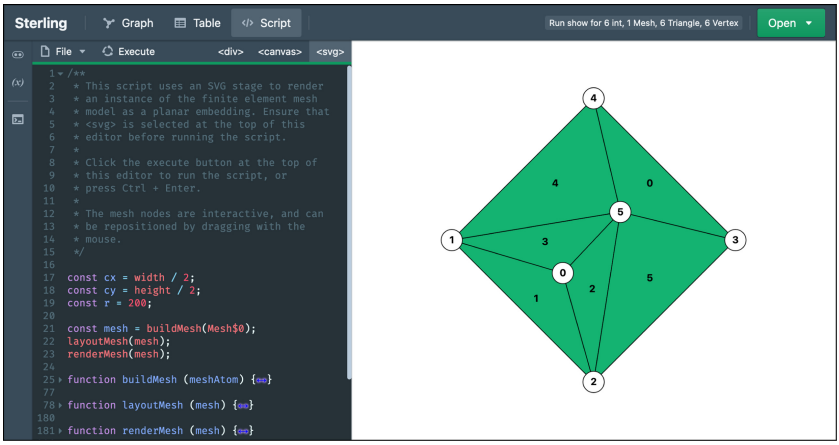
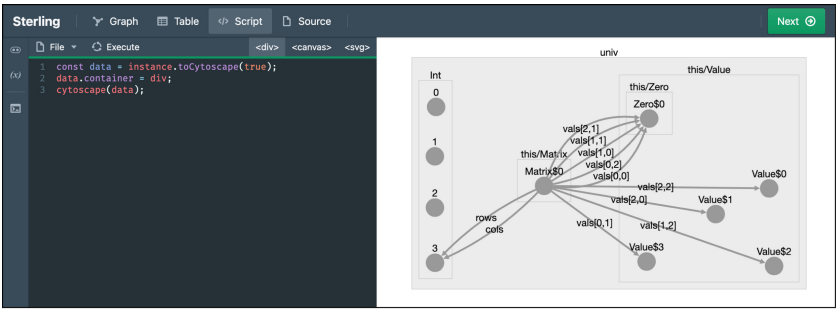


Fig. 2. Finite element mesh as a planar embedding in the script view.

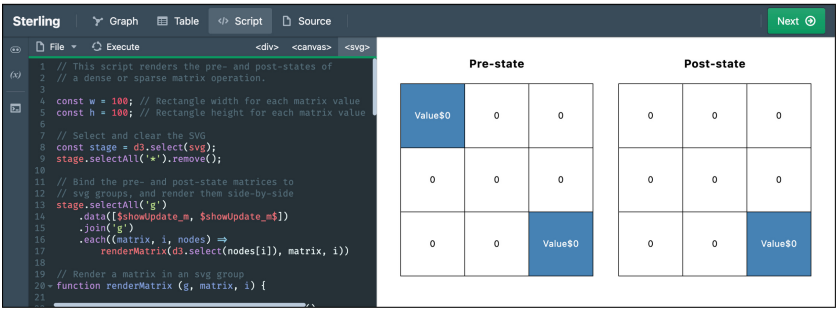
3 Creating Scripts and Models

The script view is designed to integrate with the iterative design process that is typical of Alloy, and as such, users receive the same kind of immediate visual feedback provided by the graph view, with the added benefit of complete control over the visual approach used to display instances. In typical usage of the script view, a user begins by writing a model and executing a command to generate an instance. The instance is automatically sent to Sterling, where the user then writes a script in the script view. To execute the script and generate the visualization, the user presses "Ctrl+Enter" or clicks the "Execute" button located at the top of the script editor. The user can continue to refine the visualization by editing and rerunning the script, or use the "Next" button to explore more instances. Each time an instance is generated, Sterling automatically executes the script to re-render the visualization. This automatic execution continues when the user returns to Alloy, refines the model, and generates new instances. If the model is changed in a way that causes the visualization script to throw an error, the user is notified, and they must then update the visualization script to reflect the new model.

In practice we have found visualization scripts typically start out simple and grow in complexity alongside the model. For example, early iterations of the previously described matrix models employed the Cytoscape JS library to create interactive "snapshot" views of instances, as shown in Fig. 3a. These snapshots, described by Jackson [6], proved useful in the development and understanding of both the hierarchical and relational structure of the models. As the structure of the models became more concrete and focus shifted to modeling the behavior of sparse matrix operations, the visualization script evolved to provide a more realistic view of matrices as well as a clear depiction of state change, as shown in Fig. 3b.



(a) A snapshot view of a dense matrix



(b) A sparse matrix update operation

Fig. 3. Scripts for matrix models at (a) early and (b) late stages of development.

For users who are comfortable using JavaScript, particularly those with experience using popular JS visualization libraries, creating custom visualizations from Alloy instances is straightforward. To support users with little or no experience, the Sterling website provides tutorials and numerous examples that demonstrate basic visualization techniques. Furthermore, scripts capable of generating custom visualizations for some common modeling paradigms, such as binary trees and directed graphs, are available on the website and can be used out-of-the-box.

4 Conclusion

Sterling addresses some of the common issues identified with existing visualizations in Alloy, and introduces a script view to enable development of custom visualizations without sacrificing the immediate visual feedback provided by the Alloy Analyzer. The Sterling architecture and visualization approach take inspiration from other tools developed to address certain visualization challenges in Alloy and other formalisms. Alloy4Fun [8] and BMotionWeb [7] are both web-based tools that leverage the popularity of the JavaScript programming language

and the availability of robust data visualization libraries, and PVSio-Web [10] employs a client-server architecture to enable coupling of a formal verification tool with a web-based interface. VisB [11], a tool built upon Java, JavaFX, and JavaScript, enables the creation of interactive SVG visualizations for models developed in ProB using an approach that does not require user to have prior knowledge of JavaScript.

Development is ongoing and part of the lead author's postdoc at Brown University, where Sterling's flexible architecture is being leveraged to develop user studies with the goal of better understanding the role of visualization and user interaction in state-based modeling. Additionally, Sterling is the visualizer for an Alloy-like model finder called Forge [1], which is being developed at Brown University and is used to teach a Logic for Systems class of over 60 students.

Acknowledgments. We thank Shriram Krishnamurthi, Tim Nelson, and Kathi Fisler for their ideas and support, Mia Santomauro for the Sterling custom visualization guide, and the Alloy community for their helpful suggestions. This work is partially supported by the US NSF.

References

1. Forge. https://github.com/tnelson/Forge. Accessed 12 Apr 2021
2. Baugh, J., Altuntas, A.: Formal methods and finite element analysis of hurricane storm surge: a case study in software verification. Sci. Comput. Program. **158**, 100–121 (2018)
3. Couto, R., et al.: Improving the visualization of Alloy instances. Electron. Proc. Theor. Comput. Sci. **284**, 37–52 (2018)
4. Dyer, T., Altuntas, A., Baugh, J.: Bounded verification of sparse matrix computations. In: Proceedings of the Third International Workshop on Software Correctness for HPC Applications, Correctness 2019, pp. 36–43. IEEE/ACM (2019)
5. Gammaitoni, L., Kelsen, P.: Domain-specific visualization of Alloy instances. In: Ait, A.Y., Schewe, K.D. (eds.) Abstract State Machines, Alloy, B, TLA, VDM, and Z, pp. 324–327. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-43652-3_33
6. Jackson, D.: Software Abstractions: Logic, Language, and Analysis. The MIT Press, Cambridge (2012)
7. Ladenberger, L., Leuschel, M.: BMotionWeb: a tool for rapid creation of formal prototypes. In: De Nicola, R., Kühn, E. (eds.) SEFM 2016. LNCS, vol. 9763, pp. 403–417. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-41591-8_27
8. Macedo, N., et al.: Sharing and learning Alloy on the web. arXiv abs/1907.02275 (2019)
9. Misue, K., Eades, P., Lai, W., Sugiyama, K.: Layout adjustment and the mental map. J. Vis. Lang. Comput. **6**(2), 183–210 (1995)
10. Oladimeji, P., Masci, P., Curzon, P., Thimbleby, H.: PVSio-web: a tool for rapid prototyping device user interfaces in PVS. Electron. Commun. EASST **69** (2014)
11. Werth, M., Leuschel, M.: VisB: a lightweight tool to visualize formal models with SVG graphics. In: Raschke, A., Méry, D., Houdek, F. (eds.) ABZ 2020. LNCS, vol. 12071, pp. 260–265. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-48077-6_21
12. Zaman, A., et al.: Improved visualization of relational logic models. Technical report. CS-2013-04, University of Waterloo (2013)