Multi-Way Dataflow Specifications in Graphical User Interfaces

Knut Anders Stokke[1], Mikhail Barash[1], Karl Henrik Elg Barlinn[1], Daniel Berge[1], Torjus Schaathun[1], and Jaakko Järvi[1,2]

[1] Bergen Language Design Laboratory, University of Bergen, Norway
[2] Department of Computing, University of Turku, Finland

1 Introduction

Most of us interact with graphical user interfaces (GUIs) everyday. Still, GUIs are oftentimes wrong and buggy, almost to the extent that users know to expect problems; GUIs lack necessary functionality, get user interactions wrong, and can even end up in states where the user must resort to restart the application.

The low quality of GUIs stems, at least partially, from the way GUIs are programmed today: programmers write imperative *event handlers* that are executed on every user interaction. The code in the event handlers must correctly update the different variables that comprise the GUI state while taking into account that other event handlers could be updating the GUI state simultaneously. It is common for these variables to be dependent on each other, in which case one gets into a setting of *multi-way dataflow*. While this setting is often useful to the user, and even necessary for some features, it is particularly hard to implement: a user interaction may edit different variables depending on which variables where previously edited, and event handlers must therefore keep track of the history of user events.

Consider a GUI application for scheduling talks at a conference. The GUI shows a list of talks, where each talk (`t`) has three user-editable variables: a start time (`s`), a duration (`d`), and an end time (`e`); these variables are related (`t.e = t.s + t.d`, `t.d = t.e - t.s`, `t.s = t.e - t.d`). A user can update the start time or duration of a talk and observe that the end time of the talk is updated accordingly, and that this change propagates through the succeeding talks (the latter requirement is expressed by equations $\texttt{t}_\texttt{i}\texttt{.s} = \texttt{t}_{\texttt{i-1}}\texttt{.e}$ and $\texttt{t}_\texttt{i}\texttt{.e} = \texttt{t}_{\texttt{i+1}}\texttt{.s}$). However, the GUI enables a user to update the end time of a talk too and observe that the duration or start time is updated accordingly to maintain the relation between the variables. Which of the two is updated depends on the history of actions of the user; when given a choice, the GUI should preserve the last edited variable(s). Multi-way dataflow *constraint systems* provide a solution for this setting. By using the constraint system library HotDrink [1], this example can be implemented as follows. Each talk will become a component with three variables as mentioned above, and the five equalities will become the constraints in the system. Using a planning algorithm, the library will then be able define the propagation of the data across the dependency graph.

2 Specifications in GUIs

We describe below several directions of extending the HotDrink constraint system library.

Managing structural changes. In our running example we find a structure, a list of talks, where each talk has connections to other talks through the variables that represent start times and end times. Since there are connections between elements in the structure, it is hard to manipulate the structure itself. If a GUI developer wants to insert a talk, remove a talk, or swap two talks, they must also update the connections between the talks, which is a manual and error-prone task. Many GUIs today lack basic operations to manipulate lists, and only enable users to insert and remove elements at the end of the list.

We have designed a framework to specify structures and structural manipulations on them [3]. Such specifications are used to generate functions for the operations and provide an API to the structure. This separates the concern of the lower-level details of structure manipulation from code that provides functionality to the user. Having structure specifications in a GUI could also enable external tools to better analyse the GUI code to provide additional functionality.

Spreadsheet integration. Microsoft Excel is, arguably, the most commonly used programming language; however, spreadsheet programming proves to be error-prone [2], and faulty spreadsheets may lead to poor decision making. Inserting, moving, or removing rows and columns are examples of operations that typically lead to errors: indeed, such operations must correctly update references and formulas that may range over the entire table. Traditional spreadsheet applications have no semantic understanding of how the cells relate to each other, and without any formal explanation, references between cells appear to be random. We have initiated the development of a spreadsheet application that enables the user to specify structures in a spreadsheet. The specification should enable multi-way dataflow, which to our knowledge is not possible in industry-leading tools such as Microsoft Excel or Google Sheets at the time of writing. We expect our tool to provide operations to make structural changes in the spreadsheet while keeping the structure and the dependent formulas consistent.

Graphical tool for designing GUIs. We are developing a tool to enable designers to visually specify constraints between widgets in GUIs. The tool lets designers select widgets in GUIs and specify relations between them, based on which the tool generates a constraint specification for a constraint system-based library, such as HotDrink. We plan to extend the tool to let designers visually define patterns of repeating components in GUIs, and have our tool generate a specification of feasible structural manipulations.

Macro recording and scriptable GUIs. Many of the tasks performed in GUIs are repetitive and tedious. Power users would rather automate these tasks and spend their time elsewhere. While automating repetitive command-line tasks is

often quick and easy, automating GUIs proves to be hard, time-consuming and sometimes impossible. Macro recording would enable users to record a sequence of GUI actions and later replay them over different data; the purpose is to automate oft-occurring use patterns. We are developing a generic approach for macro recording where this feature could be packaged to a library, to be reused by different GUIs. Based on these ideas, one can then generate a tailored scripting language for a particular GUI that will allow manipulating that GUI's widgets; this will enable power users to automate repetitive tasks even to a further extent.

References

1. Gabriel Foust, Jaakko Järvi, and Sean Parent. Generating reactive programs for graphical user interfaces from multi-way dataflow constraint systems. *SIGPLAN Not.*, 51(3):121–130, October 2015.
2. Raymond R Panko. What we know about spreadsheet errors. *Journal of Organizational and End User Computing (JOEUC)*, 10(2):15–21, 1998.
3. Knut Anders Stokke, Mikhail Barash, and Jaakko Järvi. Manipulating GUI structures declaratively. In *Proceedings of the 19th ACM SIGPLAN International Conference on Generative Programming: Concepts and Experiences*, GPCE 2020, page 63–69, New York, NY, USA, 2020. Association for Computing Machinery.