Adaptive GUI Layout by Satisfying Fuzzy Constraints

Takuto Yanagida
Space-Time Inc.
Sapporo, Japan
tokutadaginaya@gmail.com

Jean Vanderdonckt
Nicolas Burny
Université catholique de Louvain
Louvain Research Institute in Management and Organizations
Louvain-la-Neuve, Belgium
jean.vanderdonckt@uclouvain.be

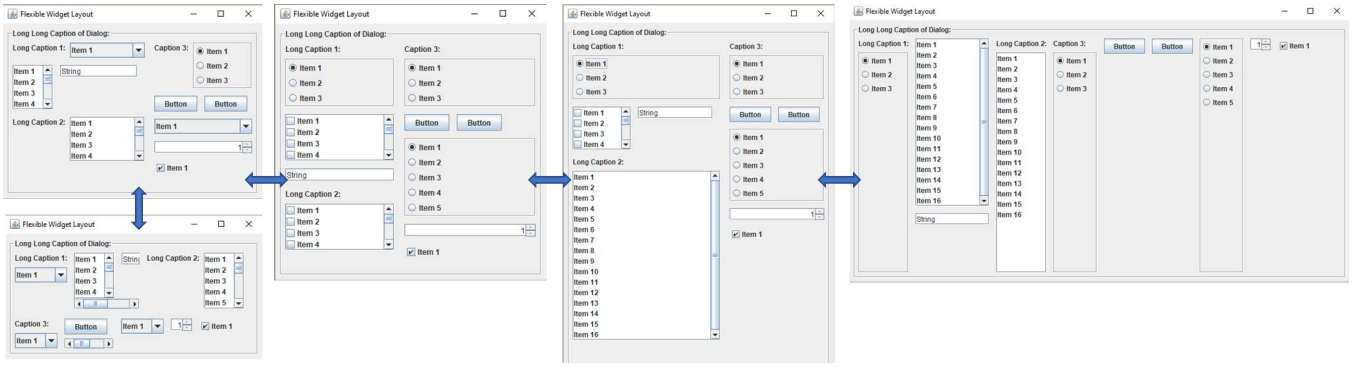

Figure 1: Adaptive GUI Layout in Java: resizing the window updates fuzzy constraints determining the widgets and their layout.

ABSTRACT

A graphical user interface is equipped with an adaptive layout when it holds the ability to dynamically adjust its layout and structure of widgets based on conditions that evolve over time coming from the end user, the platform used, and the environment. More specifically, such a layout automatically adapts itself depending on the window dimensions of the application, the screen resolution, and the screen size, thus posing a series of constraints that, sometimes, could not be entirely satisfied together at once. To cope with this variation, we formulate the adaptive GUI layout as a fuzzy constraint satisfaction problem in which the solver attempts to satisfy the most important constraints first, such as an appropriate widget selection, then the least important constraints after.

CCS CONCEPTS

• **Computing methodologies** → **Vagueness and fuzzy logic**; • **Human-centered computing** → **Graphical user interfaces**; **Web-based interaction**; • **Social and professional topics** → Software selection and adaptation; • **Software and its engineering** → *Constraint and logic languages.*

EICS '23 Companion, June 27–30, 2023, Swansea, United Kingdom

ACM ISBN 979-8-4007-0206-8/23/06.
https://doi.org/10.1145/3596454.3597190

KEYWORDS

Adaptive User Interface; GUI Layout; Plasticity of user interfaces

ACM Reference Format:
Takuto Yanagida, Jean Vanderdonckt, and Nicolas Burny. 2023. Adaptive GUI Layout by Satisfying Fuzzy Constraints. In *Companion of the ACM SIGCHI Symposium on Engineering Interactive Computing Systems (EICS '23 Companion), June 27–30, 2023, Swansea, United Kingdom.* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3596454.3597190

1 INTRODUCTION AND BACKGROUND

Adaptive layout refers to the ability of a Graphical User Interface (GUI) to dynamically adjust its layout and structure based on evolving contextual conditions [1] coming from the user, the platform, or the environment [3]. More specifically, an adaptive GUI layout automatically adapts itself to changing screen sizes, resolutions, and user preferences [6] and helps users get the best user experience by resizing the GUI to fit the available screen real estate [18]. A retrospective view of practices for supporting adaptive GUI layouts shows that they largely evolve over time: purely imperative algorithms that are hard to trace (*e.g.*, the right-bottom strategy [17]), state-space search (*e.g.*, the binary tree search [17]), rule-based systems [4] to grid-based systems [14] and constraint satisfaction problems (CSP) in forward [10] or reverse engineering [9]. Another potential practice is to design a modular GUI that can be easily rearranged at runtime by involving reusable GUI components and custom layouts that can be swapped in and out, thus enabling the layout not only to be reshuffled but also restructured and subject to widget replacement, such as in widget promotion/demotion [2]. The use of fluid layouts ensures that all elements of the design can adapt to various dimensions by changing the size and location of

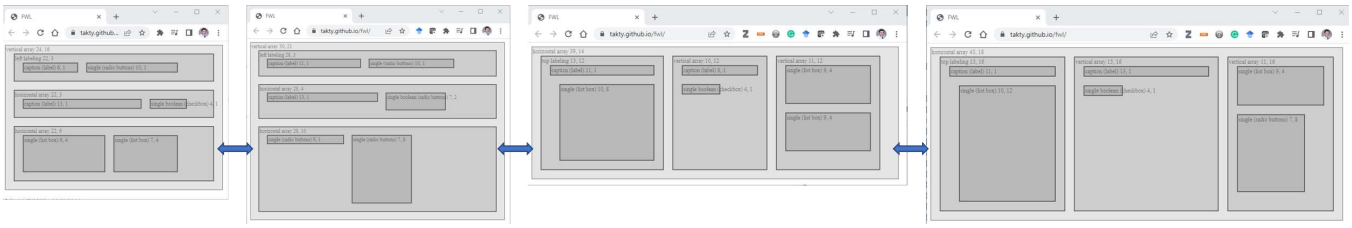

Figure 2: Adaptive GUI Layout in JavaScript: resizing the window updates constraints to locate and size its regions.

the GUI elements [19]. Frias-Martinez et al. [7] states that giving control to end users reduces the effect of incorrect adaptation at the price of an additional effort required from the users. To compare adaptivity to adaptability, they noticed that, for a digital library, users not only performed better in the adaptive GUI, but also they perceived it more positively. Users' cognitive styles have a direct impact on how users perceive adaptivity.

The 'Nokia Applet [11] expands and collapses its GUI hierarchy of widgets according to the screen sizes, resolution, and user's task: to manage a set-top box, a media player, and a smart TV. When a device is working, its GUI part is expanded in the layout, while other parts are collapsed with an animated transition [5] between. PlastiXML [4] defines the plasticity of a point as a set of (x, y) coordinates where the GUI can change according to display size and resolution constraints. Instead of solving the GUI layout problem analytically, Raneburger et al. [13] adopt a two-step heuristic approach in which GUI layout parameters are specified in a platform-independent manner via reusable transformation rules. Adaptive layout for Automotive UIs (AAUI) [8] generates GUIs for in-vehicle interaction, for which screen constraints are known at design time and, thus, do not require run-time restructuring. OR-Constraints [10] solve the adaptive GUI layout by accommodating several variations of the aspect ratio with the ORCSolver adaptive GUI renderer.

2 ADAPTIVE GUI LAYOUT BY SATISFYING FUZZY CONSTRAINTS

We formulate the problem of the adaptive GUI layout as a *fuzzy constraint satisfaction problem* (FCSP) [16], an extension of the Constraint Satisfaction Problem (CSP) already used in [9, 10]. The FCSP defines a series of fuzzy constraints, which are not necessarily entirely satisfied (since they are fuzzy), and considers the satisfaction degrees of these constraints. In the formulation, fuzzy constraints represent the mapping between widgets and desirability α as their satisfaction degrees. They enable us to represent naturally the gradual rules of the widget desirability without introducing any other objective functions. After the formulation, we can apply an FCSP solver to obtain various solutions, including suboptimal ones. FCSP solvers search combinations of assignments of variables that satisfy most constraints, but not necessarily all, with the priority given to the widget selection.

An FCSP consists of a finite set of variables $X=\{x_1, x_2, ..., x_i, ...x_n\}$, a finite set of domains of values $D=\{D_1, D_2, ..., D_i, ...D_n\}$ associated with each variable and a finite set of constraints $C = \{c_1, c_2, ..., c_j, ...c_r\}$ where a constraint c_j denotes a fuzzy relation μR_k on a subset of the variables $S_k \subset X$ [16]. S_k is called the *scope* of μR_k: a constraint is associated with its membership function, whose value is computed by an assignment to the variables in the scope of the constraint. This value is called the *satisfaction degree* as it expresses how the assignments to the variables satisfy the constraint. The minimum width of a normal widget is the sum of the width of the widest one of its items, and the width of the control parts such as a vertical scroll bar, a radio button, and a check box. The minimum height is defined by the type of widget, its item size, and the item height item. Since an FCSP requires the satisfaction of the fuzzy conjunction of all fuzzy constraints, the satisfaction degree of the whole FCSP is defined as the minimum satisfaction degree. Consequently, solving an FCSP is considered equivalent to finding the assignment yielding the best satisfaction degree of the problem from all combinations of assignments, for which various solvers exist such as JFSolver [12]. To completely address the adaptive GUI layout, constraints for selecting widgets depending on their type and size are as important as constraints for determining the location and the arrangement of these widgets to preserve their structure.

To go further in this line of research, new layout rules should be incorporated to evaluate the relation between problem scales and solving times, and to consider other FCSP algorithms. For example, constraints between sibling widgets can be added for keeping the same types and states among them. In the current implementation, we use the forward-checking algorithm, but our method is not limited to it. Formulated as FCSP, the layout problem allows us to use suboptimal results; thus, we can also consider adopting local search or hybrid algorithms such as Spread-repair-shrink (SRS) [15].

OPEN SCIENCE

Implementation details are accessible at https://takty.net/fwl/ and in the GitHub repository at https://github.com/takty/fwl. A demonstration of the version implemented in Java with JRE is accessible at https://youtu.be/Vvv1RRmZkuU and of the JavaScript version at https://takty.github.io/fwl/. We also maintain a YouTube playlist of videos demonstrating several examples of adaptive GUI layouts from the old ones to the most recent ones.

ACKNOWLEDGMENTS

The authors of this paper are very grateful to the anonymous EICS reviewers and the associate chair for their insightful and constructive comments on earlier versions of this manuscript. Jean Vanderdonckt is supported by the EU EIC Pathfinder-Awareness Inside challenge "Symbiotik" project (1 Oct. 2022-31 Dec. 2026) under Grant no. 101071147.

REFERENCES

[1] Silvia Abrahão, Emilio Insfrán, Arthur Sluÿters, and Jean Vanderdonckt. 2021. Model-based intelligent user interface adaptation: challenges and future directions. *Software and System Modelling* 20, 5 (2021), 1335–1349. https://doi.org/10.1007/s10270-021-00909-7

[2] Sara Bouzit, Gaëlle Calvary, Denis Chêne, and Jean Vanderdonckt. 2019. Interface adaptivity by widget promotion/demotion. In *Proceedings of the ACM SIGCHI Symposium on Engineering Interactive Computing Systems* (Valencia, Spain) *(EICS '19)*, José Ignacio Panach, Jean Vanderdonckt, and Oscar Pastor (Eds.). ACM, New York, NY, USA, 18:1–18:6. https://doi.org/10.1145/3319499.3328237

[3] Gaëlle Calvary, Joëlle Coutaz, David Thevenin, Quentin Limbourg, Laurent Bouillon, and Jean Vanderdonckt. 2003. A Unifying Reference Framework for multi-target user interfaces. *Interacting with Computers* 15, 3 (2003), 289–308. https://doi.org/10.1016/S0953-5438(03)00010-9

[4] Benoît Collignon, Jean Vanderdonckt, and Gaëlle Calvary. 2008. Model-Driven Engineering of Multi-target Plastic User Interfaces. In *Proceedings of Fourth IEEE International Conference on Autonomic and Autonomous Systems* (Gosier, Guadeloupe) *(ICAS '08)*. IEEE Computer Society, Los Alamitos, USA, 7–14. https://doi.org/10.1109/ICAS.2008.37

[5] Charles-Eric Dessart, Vivian Genaro Motti, and Jean Vanderdonckt. 2011. Showing User Interface Adaptivity by Animated Transitions. In *Proceedings of the 3rd ACM SIGCHI Symposium on Engineering Interactive Computing Systems* (Pisa, Italy) *(EICS '11)*. Association for Computing Machinery, New York, NY, USA, 95–104. https://doi.org/10.1145/1996461.1996501

[6] Leah Findlater and Krzysztof Z. Gajos. 2009. Design Space and Evaluation Challenges of Adaptive Graphical User Interfaces. *AI Magazine* 30, 4 (2009), 68–73. https://doi.org/10.1609/aimag.v30i4.2268

[7] Enrique Frias-Martinez, Sherry Y. Chen, and Xiaohui Liu. 2009. Evaluation of a personalized digital library based on cognitive styles: Adaptivity vs. adaptability. *International Journal of Information Management* 29, 1 (2009), 48–56. https://doi.org/10.1016/j.ijinfomgt.2008.01.012

[8] Renate Häuslschmid, Klaus Bengler, and Cristina Olaverri-Monreal. 2013. Graphic Toolkit for Adaptive Layouts in In-Vehicle User Interfaces. In *Proceedings of the 5th International Conference on Automotive User Interfaces and Interactive Vehicular Applications* (Eindhoven, Netherlands) *(AutomotiveUI '13)*. Association for Computing Machinery, New York, NY, USA, 292–298. https://doi.org/10.1145/2516540.2516580

[9] Yue Jiang, Wolfgang Stuerzlinger, and Christof Lutteroth. 2021. ReverseORC: Reverse Engineering of Resizable User Interface Layouts with OR-Constraints. In *Proceedings of ACM Int. Conference on Human Factors in Computing Systems* (Yokohama, Japan) *(CHI '21)*. ACM, 316:1–316:18. https://doi.org/10.1145/3411764.3445043

[10] Yue Jiang, Wolfgang Stuerzlinger, Matthias Zwicker, and Christof Lutteroth. 2020. ORCSolver: An Efficient Solver for Adaptive GUI Layout with OR-Constraints. In *Proceedings of ACM International Conference on Human Factors in Computing Systems* (Honolulu, HI, USA) *(CHI '20)*. ACM, 1–14. https://doi.org/10.1145/3313831.3376610

[11] Heikki Keränen and Johan Plomp. 2002. Adaptive runtime layout of hierarchical UI components. In *Proceedings of the Second Nordic Conference on Human-Computer Interaction* (Aarhus, Denmark) *(NordiCHI '02)*, Olav W. Bertelsen (Ed.). ACM, 251–254. https://doi.org/10.1145/572020.572058

[12] Ryszard Kowalczyk and V. Bui. 2001. JFSolver: a tool for modeling and solving fuzzy constraint satisfaction problems. In *Proceedings of 10th IEEE International Conference on Fuzzy Systems*, Vol. 1. 304–307. https://doi.org/10.1109/FUZZ.2001.1007309

[13] David Raneburger, Roman Popp, and Jean Vanderdonckt. 2012. An automated layout approach for model-driven WIMP-UI generation. In *Proceedings of the ACM SIGCHI Symposium on Engineering Interactive Computing Systems* (Copenhagen, Denmark) *(EICS '12)*, Simone Diniz Junqueira Barbosa, José Creissac Campos, Rick Kazman, Philippe A. Palanque, Michael D. Harrison, and Steve Reeves (Eds.). ACM, New York, NY, USA, 91–100. https://doi.org/10.1145/2305484.2305501

[14] Morteza Shiripour, Niraj Ramesh Dayama, and Antti Oulasvirta. 2021. Grid-Based Genetic Operators for Graphical Layout Generation. *Proc. ACM Hum.-Comput. Interact.* 5, EICS, Article 208 (may 2021), 30 pages. https://doi.org/10.1145/3461730

[15] Yasuhiro Sudo and Masahito Kurihara. 2006. Spread-Repair-Shrink: A Hybrid Algorithm for Solving Fuzzy Constraint Satisfaction Problems. In *Proceedings of the IEEE International Conference on Fuzzy Systems* (Vancouver, BC, Canada) *(FUZZY 2006)*. IEEE Computer Society Press, Piscataway, NJ, USA, 2127–2133. https://doi.org/10.1109/FUZZY.2006.1681995

[16] Yasuhiro Sudo, Masahito Kurihara, and Tamotsu Mitamura. 2006. Extending Fuzzy Constraint Satisfaction Problems. *Journal of Advanced Computational Intelligence and Intelligent Informatics* 10, 4 (2006), 465–471. https://doi.org/10.20965/jaciii.2006.p0465

[17] Jean Vanderdonckt, Missiri Ouedraogo, and Banta Ygueitengar. 1994. A Comparison of Placement Strategies for Effective Visual Design. In *Proceedings of BCS Int. Conf. on Human-Computer Interaction, People and Computers IX* (Glasgow, UK) *(HCI '94)*, Gilbert Cockton, Stephen W. Draper, and George R. S. Weir (Eds.). Cambridge University Press, 125–143.

[18] Rares Vernica and Niranjan Damera Venkata. 2015. AERO: An Extensible Framework for Adaptive Web Layout Synthesis. In *Proceedings of the 2015 ACM Symposium on Document Engineering* (Lausanne, Switzerland) *(DocEng '15)*. Association for Computing Machinery, New York, NY, USA, 187–190. https://doi.org/10.1145/2682571.2797084

[19] Clemens Zeidler, Christof Lutteroth, Wolfgang Stürzlinger, and Gerald Weber. 2013. The Auckland layout editor: an improved GUI layout specification process. In *Proceedings of The 26th Annual ACM Symposium on User Interface Software and Technology* (St. Andrews, United Kingdom) *(UIST '13)*, Shahram Izadi, Aaron J. Quigley, Ivan Poupyrev, and Takeo Igarashi (Eds.). ACM, 343–352. https://doi.org/10.1145/2501988.2502007

,