(Semi-)Automatic Computation of User Interface Consistency

Nicolas Burny
Université catholique de Louvain
LouRIM
Louvain-la-Neuve, Belgium
nicolas.burny@uclouvain.be

Jean Vanderdonckt
Université catholique de Louvain
LouRIM
Louvain-la-Neuve, Belgium
jean.vanderdonckt@uclouvain.be

ABSTRACT

Many measures exist to (semi-)automatically compute the quality of a graphical user interface, such as aesthetic metrics, visual metrics, and performance metrics. These measures are mostly individual as they apply to a single graphical user interface at a time. Unlike these measures, consistency requires evaluating a number of screens within the same application (intra-application consistency) or across applications (inter-application consistency). This paper presents a formula, a method, and supporting software for computing this consistency and its counterpart, inconsistency, either completely automatically when the interface segmentation is performed by the software or semi-automatically when the interface segmentation is performed manually by the end-user.

CCS CONCEPTS

• **Human-centered computing** → *Usability testing*; **Graphical user interfaces**; **Web-based interaction**; Systems and tools for interaction design.

KEYWORDS

Aesthetics, automatic evaluation, consistency, measurement, visual design, visual metrics.

ACM Reference Format:
Nicolas Burny and Jean Vanderdonckt. 2022. (Semi-)Automatic Computation of User Interface Consistency. In *Companion of the 2022 ACM SIGCHI Symposium on Engineering Interactive Computing Systems (EICS '22 Companion), June 21–24, 2022, Sophia Antipolis, France.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3531706.3536448

1 INTRODUCTION

The concept of consistency of Graphical User Interfaces (GUIs) knows a great variation of definitions. For example, Scapin & Bastien [4] define it as "The criterion Consistency refers to the way interface design choices (codes, naming, formats, procedures, etc.) are maintained in similar contexts and are different when applied to different contexts". Shneiderman *et al.* [27] argue that "consistency has no meaning on its own; it is inherently a relational concept. Therefore to merely say that an interface is consistent or that consistency is a goal of user interface design".

The rationale behind consistency is that all GUI elements, such as menus and their submenus, commands, labels, fields, controls, icons, and images, will be better recognized and used if their location, dimensions, arrangement, and format remain the same from one screen to the other with the same GUI (*intra-application consistency* [22] or *internal consistency* [13]) or from one GUI to the next (*inter-application consistency* [26] or *external consistency* [13]). A third type of consistency is also defined [13]: *external analogical or metaphorical consistency* refers to consistency between the GUI and the features of the domain, in particular its concepts, their representations, and related metaphors. Zuschlag [35] indeed argues that all GUI elements are concerned: symbols (imagery that conveys information), codes (*e.g.*, colors, sizes, weights, marks, metaphor), units of measurement, data formats, abbreviations (*e.g.*, terms, function names, control keys, labels), layouts (GUI elements' locations on a page or in a menu, window, or temporal sequence, navigation). More specifically, Wangmi [33] studied which GUI elements are expected to remain consistent across multiple screens like on desktop, tablet, smart phone, and smart television.

A consistent GUI is more predictable, more easy to learn [26], and to reuse (according to the principle of "learn once, use many", thus improving the completion time and rate of tasks [1] and increasing subjective satisfaction [25], the three main factors in usability. An inconsistent GUI deteriorates user performance [14], the visual search time [18], the error rate, and induces user rejection [23]. Although the concept of consistency is correctly understood by most stakeholders, its assessment remains an open debate, and methods vary depending on the GUI elements for which consistency should be checked [16]: comparing the consistency of icon graphics, text meaning, etc. requires a semantic understanding of these elements, which is difficult to achieve [5], but checking the consistency of the layout is more achievable. syntactically.

This paper focuses on the latter approach by hypothesizing that visual design measures, if they are consistently applied and measured on a series of screens, should represent an admissible indicator of the visual consistency of these screens. After reviewing previous work on GUI consistency with a focus on software support (Section 2), we provide our definition of GUI consistency based on visual design measures (Section 3) as software computable (Section 4). In this last section, we describe our implementation for computing this consistency and illustrate its application in various conditions: automatic computation with (semi)automatic segmentation, intra- *vs.* inter-application, and custom evaluation by measure selection.

EICS '22 Companion, June 21–24, 2022, Sophia Antipolis, France

ACM ISBN 978-1-4503-9031-6/22/06…$15.00
https://doi.org/10.1145/3531706.3536448

2 RELATED WORK

Sherlock [17] evaluates the consistent usage of fonts, widget sizes, and colors of an MS Windows GUI based on its resource files, making it specific to this operating system, but effective since all accessible resources can be syntactically compared. For example, push button analysis detects variations in labels, typefaces, colors, sizes, and placement. Although consistent alignment and colorimetry can be effectively checked, other visual measures, such as regularity and density, are left out. Sierra [31] enables a designer to search for usability guidelines related to consistency, even with fine-tuned selection. For example, a simple query on consistency guidelines performed with Sierra on Smith & Mosier [29] usability guidelines returns more than 60 guidelines. But no practical method on how to evaluate these guidelines is provided. Kwaresmi [6] automatically parses the HTML code of a web site to verify usability guidelines encoded in Guideline Definition Language (GDL), a domain-specific language for usability guidelines. However, it does not actually contain any guideline to verify consistency, although it could if an engineer effectively creates the corresponding rules, which requires modeling abilities.

Consistency could also be ensured by construction of model-based approaches for automatic GUI generation [2], such as transformation templates [3], for multiple user interfaces [11] and multi-device interfaces [19], to solve the mapping problem [9]. For example, Uniform [21] automatically generates consistent GUIs for remote controls: the same rules are used to generate the same controls for different versions of a GUI. Jelly [19] does a similar job for multi-device GUIs to ensure the inter-device consistency for the same application. The consistency of the GUI could also be addressed by action analysis [12], which assigns a weight to each element of the GUI specified on the screens of a single GUI or in different adaptations of the same GUI. After noticing that the consistency of the GUI is better covered and studied in design than in evaluation, Chen *et al.* [8] proposed using incomplete matching tasks and paired comparison methods to detect and analyze variations between the various GUIs. These methods can identify the performance of consistency between different GUIs and be able to find out some GUI elements deteriorating the overall consistency.

More recently, Silva & Winckler [28] propose a scenario-based approach for checking consistency in GUI design artifacts, modeling business, and user requirements. In their work, consistency is considered both at the GUI level, and between the business models and the task models. This touches the field of model consistency which is outside the scope of this paper. Silva *et al.* [24] ensure consistency between user requirements and modeling artifacts by relying on a Behavior-Driven Development (BDD) to provide automated assessment for task models, which model the flow of user and system tasks in the interactive software. Li *et al.* [15] present a model-based approach exploiting a Petri net to identify GUI inconsistency. Consistency rules exist for various operations using XML protocol with the aim at achieving standard GUI operations. Tudor & Coyle [30] report that key ingredients for maintaining consistency are: an effective process, active participation of stakeholders, iterative design and identical review cycles, and reusable code. In conclusion, consistency checking is either limited to system-dependent resources or addressed in model-based GUI design.

3 A COMPUTATIONAL DEFINITION OF GUI CONSISTENCY

If we had to evaluate the consistency between different screens of an interface, we would in principle have to identify all the elements that make it up, obtain all the properties that govern them, and check all the variations from one screen to another. Unfortunately, the number of different elements, their different types, the large number of possible properties per element and the size of the domain of possible values for each property make solving this problem intractable. The combination of all these possibilities would produce a non-deterministic model, subject to a combinatorial explosion.

On the other hand, a deterministic model can be obtained by aggregating all a series of individual properties (*e.g.*, position, dimensions, arrangement) by means of global measures (*e.g.*, alignment based on positions, regularity based on dimensions, and proportion based on arrangement) for which there are computable formulas. As a matter of fact, these visual techniques [32] could be associated to calculable aesthetic metrics [34]. If these metrics are calculated on all the screens on which consistency applies, then any significant variation in these metrics should indicate a source of inconsistency. The values of these metrics are then compared to find discrepancies such as outliers and could be done generally by calculating the standard deviation between them. Therefore, our computable definition of GUI inconsistency consists of calculating a set of metrics for a number of screens, calculating their standard deviation with respect to the average value, normalizing the results, and computing the consistency as the complementary as follows:

$$Inconsistency = \frac{1}{N} \sum_{n \in N} \frac{1}{Xmax_n} \sqrt{\frac{1}{J-1} \sum_{j \in J} (Xj_n - \overline{Xn})^2} \quad (1)$$

$$Consistency = 1 - Inconsistency \quad (2)$$

where
N = the number of metrics per screen.
Xs_n = the value X computed for metric n on screen s.
$\overline{Xn}$ = the mean value for metric n over all screens.
$Xmax_n$ = the maximum value for a metric n over all screens.
J = the number of screens of the GUI to evaluate.

4 SOFTWARE FOR (SEMI-)AUTOMATIC COMPUTATION OF GUI CONSISTENCY

"GraphicaL User COnSistency Evaluator" (Glucose) is an on-line web application for computing inconsistency (Eq. 1) and consistency (Eq. 2) of a GUI composed of a set of screen images (*e.g.*, a screenshot, a picture, a sketch, or a static prototype). Each *image* is decomposed into *regions* (with id, location, dimensions), surrounded by *frames* (with size, border, width, depth) and composed of *pixels* (with coordinates, color).

4.1 Implementation

Initially, we developed Glucose in the form of microservices to make them easily extensible and interoperable in other software, like uiLab [7]. This approach requires the identification of a significant common part between microservices, which was not possible due to the independence of the metrics. Therefore, we decided to develop Glucose within PyCharm, an Integrated Development Environment (IDE) specifically designed for programming in Python

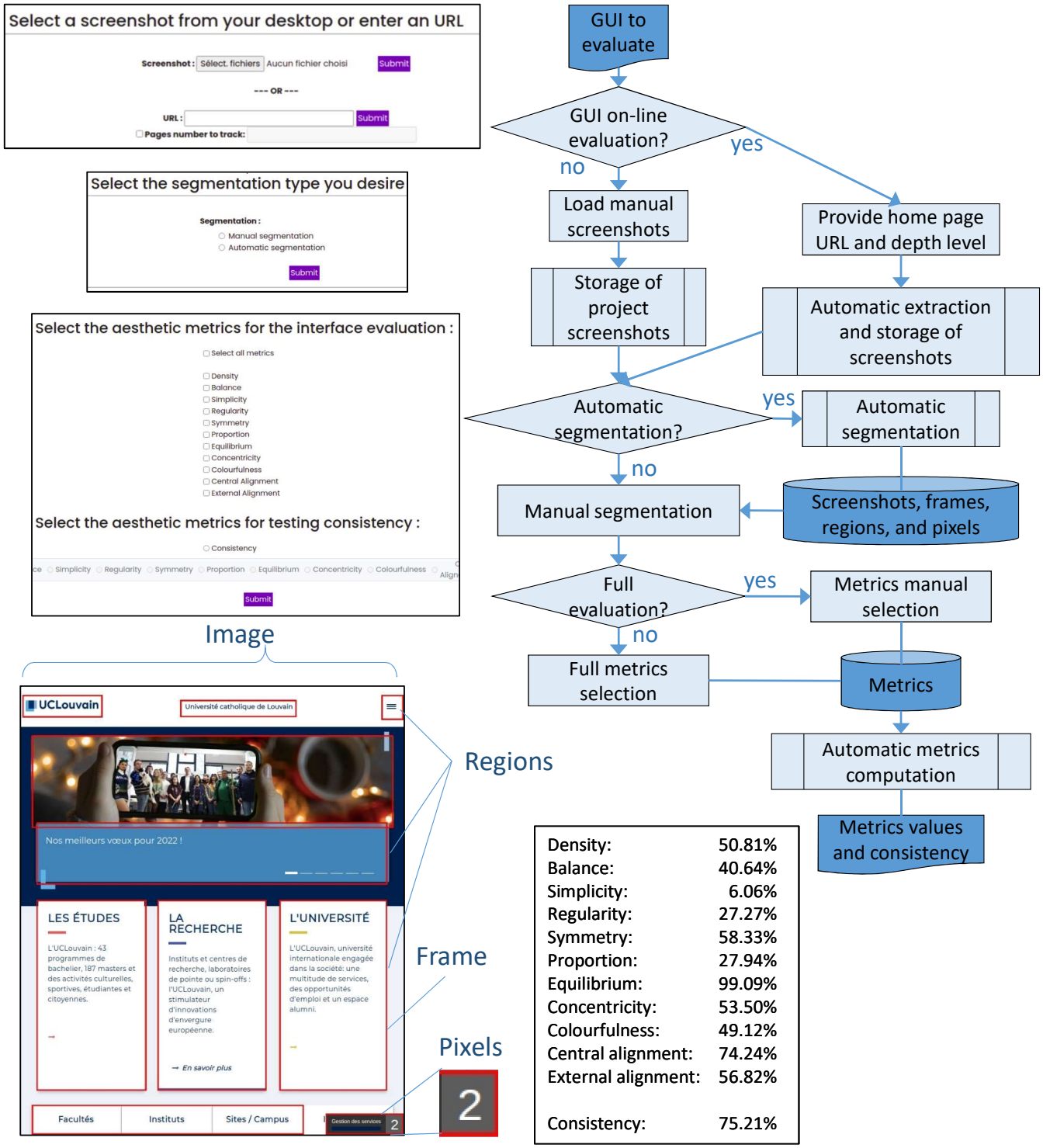


Figure 1: Method for computing GUI consistency.

language, more particularly with Flask, a lightweight micro-web framework written in Python that does not require particular tools or libraries to ensure the front-end. GUIs, their images, regions, frames, and pixels are stored in a Sqlite relational database that allows data storage and management using the SQL language. It is also very flexible to interact with the backend because Flask provides adequate libraries to manage the interaction with this type of database. We chose Python to facilitate the programming of metrics based on GUIs, images, regions, frames, and pixels. Each metric is computed inside a eponym subclass of the superclass Metric.

To this day, eleven metrics are implemented based on the formula found in the literature [1, 17, 20, 23, 34]: Balance represents the image balance computation using collected data from regions and frames, Central alignment represents the image central alignment computation using collected data from regions, Colourfulness represents the image colourfulness computation using collected data from pixels, Concentricity represents the image concentricity computation using collected data from regions and frames, Density represents the image density computation using collected data from regions and frames, etc. for Equilibrium, External alignment, Proportion, Regularity, Simplicity and Symmetry. New metrics could be added by creating another subclass of Metric. For example, the algorithm for computing density is based on the formula proposed by Ngo *et al.* [20]. Since the density can be considered as the extent to which the screen is covered with GUI elements, it can be measured as the percentage of the entire frame covered with GUI elements, as follows:

$$Area = \sum_{r \in R} r_{width} \times r_{height} \tag{3}$$

$$\text{Density} = \frac{Area}{frame_{width} \times frame_{height}} \tag{4}$$

where
R = the image regions segmented by the image processing .
r_{width} = the width of the region.
r_{height} = the height of the region.
$frame_{width}$ = the width of the GUI.
$frame_{height}$ = the height of the GUI.

The algorithm for computing concentricity calculates the average of the distances between the GUI barycenter and the center of its elements [34]:

$$frame_{diagonal} = \sqrt{frame_{width}^2 \times frame_{height}^2} \tag{5}$$

$$Distances = \sum_{r \in R} \sqrt{\left(r_{ctr-x} - \frac{frame_{width}}{2}\right)^2 + \left(r_{ctr-y} - \frac{frame_{height}}{2}\right)^2} \tag{6}$$

$$\text{Concentricity} = \frac{2 \times Distances}{n \times frame_{diagonal}} \tag{7}$$

where
R = the image regions segmented by the image processing .
n = the total number of the GUI regions.
$frame_{width}$ = the width of the GUI.
$frame_{height}$ = the height of the GUI.
r_{ctr-x} , r_{ctr-y} = the coordinates of the center of the region.

The algorithm to compute the central alignment is based on [34]:

$$\text{Central Alignment} = \frac{CAV + CAH}{4n} \tag{8}$$

where
n = the total number of the GUI regions.
CAV = number of center aligned GUI elements vertically.
CAH = number of center aligned GUI elements horizontally.
Similarly, the external alignment can be computed as follows:

$$\text{External Alignment} = \frac{DAV + DAH}{4n} \tag{9}$$

where
n = the total number of the GUI regions.
DAV = number of vertically aligned segments.
DAH = number of horizontally aligned segments.

4.2 Method

The method for evaluating the consistency of a GUI consists of the following stages (Fig. 1): the user can either upload a set of manually taken screenshots (or pictures, sketches, provided that they belong to the same style- sketches could not be mixed up with pictures) or provide the URL of a home page of a web site, along with a *depth level* specifying how many web pages will be automatically retrieved from the home page. For example, a depth level of five will create five screenshots: one of the home page in the current resolution and four other by recursively examining the HTML code to find out other pages of the same site. Screenshots are then stored in a project database. The user then selects the desired segmentation type: *automatic* (which automatically detects regions and frames using edge detection, image, and text recognition techniques) or *manual* (which allows the user to draw contours from any region of interest). Automatic segmentation could also be interactively reviewed later, and several segmentations can be created for a project, manual and automatic. The user selects the metrics involved in the computation. By default, a full evaluation computes all metrics, all average values of these metrics for all screenshots of the project. Metrics involved in the consistency computation can also be (un)selected. The metrics and consistency values are displayed after automatic computation. Appendix A provides a detailed representation of these processes based on UML Activity diagrams. Fig. 6 represents the activity of evaluating the screenshots of a GUI with manual segmentation, while Fig. 7 represents its counterpart activity where GUI screenshots are automatically segmented and interactively refined if needed. Fig. 8 and Fig. 9 represent the activity of evaluating a GUI by providing its URL and a depth level by manual segmentation, respectively, by automated segmentation.

For example, Fig. 1, bottom, reproduces a screenshot obtained with automatic segmentation and results in all metrics computed and a consistency rate of 75.21% with the other screenshots of the project. Of course, this rate can change depending on the screenshots stored in the project and can be re-computed at any time, thus enabling the user to compare various consistency configurations. In this way, intra-application consistency is computed by providing all screenshots of interest of the same application or web site and inter-application is obtained similarly by providing screenshots of the various applications. Different versions of the same application can be compared in this way. Now we present some use cases.

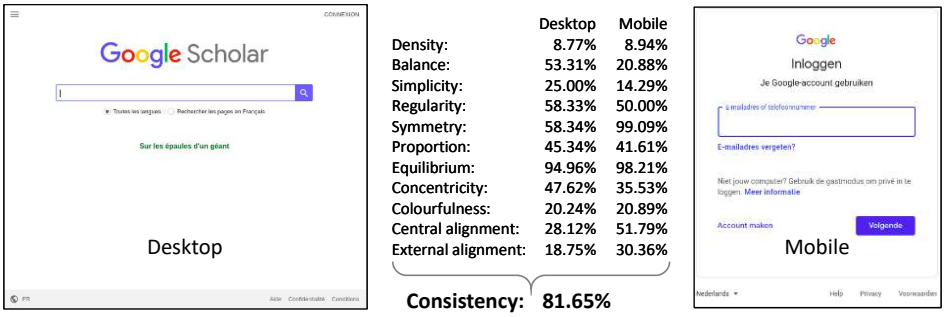


Figure 2: Example of GUI consistency computation between a desktop and a mobile.

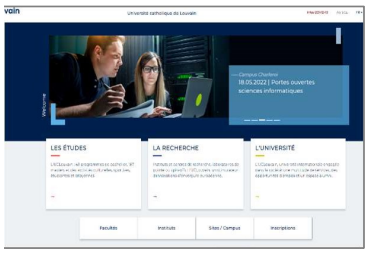
Inter-platform consistency. When a website is declined in different versions with different presentations between platforms, it can be interesting to estimate the consistency. For example, Fig. 2 reproduces the results obtained for all metrics and consistency for the Google Scholar home page captured on a desktop (left) and on a mobile device (right), obtaining a consistency rate of 81.65% between them. Google home pages are recognized for their very low screen density, which is confirmed by a screen density of 8.77% for the desktop and 8.94% for the mobile device, which are pretty similar. Their colourfulness is also estimated in a very similar way: 20.24% for the desktop version *vs.* 20.89% for the mobile version.

Inter-application, intra-domain consistency. When two or more applications belong to the same field of human activity, they often try to differentiate from each other to make themselves more competitive. In this case, the owner of an application is interested in knowing to what extent the interface of this application GUI resembles those of competitors or to what extent this GUI differs from others. Or, in contrast, one wants to evaluate to what extent the GUIs of two websites belonging to the same domain of human activity are similar. For example, Fig. 3 reports a consistency of 76% between the home page of "Université catholique de Louvain (UCLouvain)" and the home page of "Katholieke Universiteit Leuven (KULeuven)", two sisters universities speaking French and Flemish, respectively. Although the contents of the two screenshots belong to the same academic domain, they are still different, but are presented in a similar way to some extent. The density of the left screenshot (UCLouvain, 54%) is estimated lighter than the right screenshot (KULeuven, 98%) since the first contains some whte areas while the second is all coloured, even in very light colours. The balance is also significantly different: 2% *vs.* 20%. However, concentricity is estimated similar: 77% *vs.* 71%.

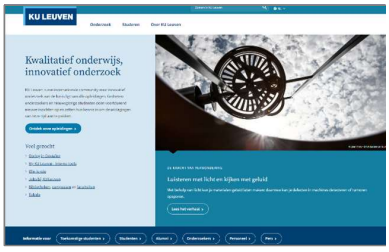
Inter-context consistency. When an application is sensitive to its context of use, its interface may vary depending on conditions originating from the user (*e.g.*, the profile), from the platform (*e.g.*, a smartphone), and from the physical environment (*e.g.*, the location, the circumstances). Under these conditions, it becomes interesting to show how the interface adapts over time and to what extent it remains consistent with itself [10]. For example, Fig. 4 evaluates the consistency of the Walkaware application, a context-sensitive web application for trip planning synchronized with geolocation and environmental conditions. More specifically, Fig. 4 considers the application screenshot captured in the browser of a car with daylight (left) *vs.* under night conditions (right). The consistency between these two screenshots is calculated as 63%. Semantically speaking, the GUI elements of both interfaces remain the same: the URL bar, the road map, the fields for queries, the push button, etc. Consequently, measures related to location remain comparable, such as symmetry, proportion, and alignment. However, their contents change since the context evolved between the two screenshots and the colorimetry is largely affected. The density is estimated similar: 85% for the daylight GUI *vs.* 93% for the night version. The dark background of the night version causes some problems with the automatic segmentation of the regions, thereby explaining the 100% for simplicity and regularity.

Inter-sketch consistency. When in the early stages of GUI design, participatory design recommends prototyping the interface, for example using low-fidelity sketches. Before moving to higher fidelity in development, it is important to ensure consistency between these sketches. For example, Fig. 5 evaluates the consistency between two sketches of a smartphone application to rent a car. Since sketches, by definition, invoke only a very limited number of colors, measurements of colorimetry return low percentages, such as 16% and 14% in our case. The automatic segmentation of regions induces some limitations for such screenshots. For example, the density is estimated to be very high (93% and 98%) because detected regions, such as group boxes, tabbed dialog boxes, and navigation bars are assumed to consume their entire space, not counting their interior density. Similarly, simplicity, regularity, and symmetry reach their maximum values because the detected regions cover almost the entire surface of the smartphone. Consequently, alignment measures, such as central alignment and external alignment, are estimated to be poor, as the first-level regions are not aligned. In the case of these sketches, there is a need to perform a recursive computation of the measures not only at the first-level of regions, but also at the second level.

In conclusion, consistency can also be evaluated for different devices, computing platforms, and various configurations (*e.g.*, different screen resolutions and orientations for a smartphone). The manual loading of screenshots enables computing the consistency between different versions of the same application for various contexts of use, *i.e.*, different users (*e.g.*, comparing the general purpose home page of Google Scholar with one obtained after logging on), different platforms (*e.g.*, comparing the same web page rendered on different screens with varying resolutions or comparing different implementations of the same application for different operating systems), and different environments (*e.g.*, comparing a screenshot in normal lighting conditions *vs.* dark conditions).

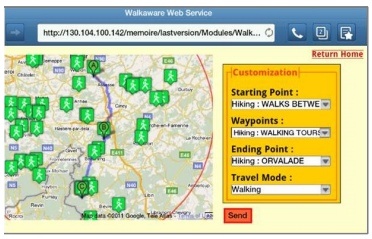


	Left (UCL)	Right (KUL)
Density:	54.00%	98.07%
Balance:	2.2%	20.09%
Simplicity:	8.33%	5.41%
Regularity:	34.72%	44.14%
Symmetry:	58.33%	58.33%
Proportion:	25.66%	55.74%
Equilibrium:	96.25%	98.67%
Concentricity:	77.80%	71.29%
Colourfulness:	44.42%	70.01%
Central alignment:	59.38%	45.95%
External alignment:	52.08%	24.32%

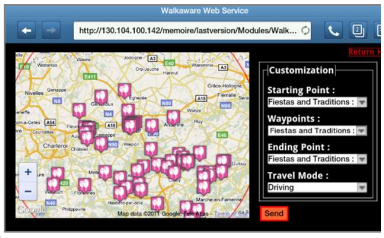


Consistency: 76.27%

Figure 3: Example of GUI consistency computation between two university home pages.

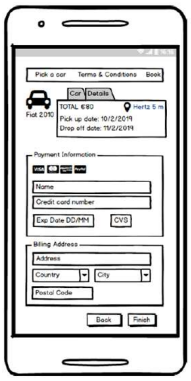


	Day	Night
Density:	85.82%	93.17%
Balance:	60.59%	30.06%
Simplicity:	20.01%	100.00%
Regularity:	56.67%	100.00%
Symmetry:	58.33%	59.09%
Proportion:	76.27%	93.71%
Equilibrium:	94.54%	45.97%
Concentricity:	68.05%	48.14%
Colourfulness:	100.00%	51.63%
Central alignment:	20.10%	20.10%
External alignment:	17.50%	17.50%



Consistency: 63.30%

Figure 4: Example of GUI consistency for a context-aware application.



	Left (Payment)	Right
Density:	98.38%	98.07%
Balance:	0.0%	0.0%
Simplicity:	100.00%	100.00%
Regularity:	100.00%	100.00%
Symmetry:	100.00%	100.00%
Proportion:	99.67%	99.01%
Equilibrium:	23.04%	24.59%
Concentricity:	64.44%	63.49%
Colourfulness:	16.05%	14.83%
Central alignment:	0.00%	0.00%
External alignment:	0.00%	0.00%

Consistency: 79.66%

Figure 5: Example of GUI consistency computation between two GUI sketches.

5 BENEFITS AND LIMITATIONS

Glucose adopts a pixel-based approach to compute its metrics based on images, regions, frames, and pixels. Consequently, this syntactical process is completely agnostic of semantic aspects (*e.g.*, text meaning, color metaphors, familiar conventions), dynamic aspects (*e.g.*, navigation objects, menu structure), and of interaction techniques and devices (*e.g.*, mouse cursor, touch, pointing). Among the eleven metrics implemented so far, Central Alignment, External Alignment, Concentricity, and Simplicity are metrics oriented towards assessing the consistent location of GUI elements based on their coordinates. Balance, Symmetry, Equilibrium, and Regularity are metrics oriented towards assessing the consistent distribution of GUI elements. Proportion assesses the consistent sizes among GUI elements, whatever they are. Density is the only metric that assesses the consistent "filling" of the GUI in terms of its elements, but it does not know anything about these elements. Consequently, two GUIs could share the same approximate density while having completely different GUI elements located at different positions. This is why metrics are complementary and why the overall consistency formula could be computed on any set of metrics, preferably a larger set of individual metrics. Similarly, Colourfulness assesses the consistent usage of colors but two GUIs could converge more or less towards the same colourfulness while having completely different GUI elements. Estimating the nature of the GUI elements is not supported currently and should be based on image recognition and text recognition techniques. Otherwise, the consistency will always remain assessed at a syntactical level. For example, DUAL [5] automates the computation of several GUI metrics based on an ontology and recognizes that

terminological consistency is very challenging to obtain and, therefore, excludes it. Its major interest lies in its automatic detection of any pixel variation that causes interference to consistency. The inconsistency formula then cumulates the interference detections based on the metrics calculated on demand (all of them or a subset). Therefore, the expression of consistency depends on the ability of the selected metrics to account for the aspects involved in the GUI layout. This expression is not necessarily aligned with the human interpretation of consistency since it varies widely, but it allows to be systematic and reproducible.

6 CONCLUSION

This paper presents Glucose, a software that automatically calculates 11 metrics in a GUI to deduce its inconsistency and consistency rate. The GUI consists of different screens of variable format, variable type, and from different sources, thereby supporting consistency checking in multiple configurations, such as, but not limited to: intra-application, inter-application, inter-device/platform, multi-resolution, multiple variants. Future work will cover account management to manage large projects by sending them by batches for consistency evaluation to a cloud server.

REFERENCES

[1] Ahamed AlTaboli and Mohammad Raafat Abou-Zeid. 2007. Effect of Physical Consistency of Web Interface Design on Users' Performance and Satisfaction. In *Proc. of HCI Int. '07*. 849–858. https://doi.org/10.1007/978-3-540-73111-5_94

[2] Anthony Anjorin, Enes Yigitbas, Hermann Kaindl, and Roman Popp. 2018. On the Development of Consistent User Interfaces (Extended Abstract) *(Programming'18 Companion)*. 18–20. https://doi.org/10.1145/3191697.3191716

[3] Nathalie Aquino, Jean Vanderdonckt, and Oscar Pastor. 2010. Transformation Templates: Adding Flexibility to Model-Driven Engineering of User Interfaces *(SAC '10)*. 1195–1202. https://doi.org/10.1145/1774088.1774340

[4] J. M. Christian Bastien and Dominique L. Scapin. 1995. Evaluating a user interface with ergonomic criteria. *Int. J. Hum. Comput. Interact.* 7, 2 (1995), 105–121. https://doi.org/10.1080/10447319509526114

[5] Michaela Bačíková and Martin Zbuška. 2015. Towards automated evaluation of domain usability. In *Proceedings of IEEE 13th International Scientific Conference on Informatics* (Poprad, Slovakia) *(ISCI '15)*. IEEE Computer Society Press, Los Alamitos, CA, USA, 41–46. https://doi.org/10.1109/Informatics.2015.7377805

[6] Abdo Beirekdar, Jean Vanderdonckt, and Monique Noirhomme-Fraiture. 2002. *A Framework and a Language for Usability Automatic Evaluation of Web Sites by Static Analysis of HTML Source Code.* Springer Netherlands, Dordrecht, 337–348. https://doi.org/10.1007/978-94-010-0421-3_29

[7] Nicolas Burny and Jean Vanderdonckt. 2021. UiLab, a Workbench for Conducting and Reproducing Experiments in GUI Visual Design. *Proc. ACM Hum. Comput. Interact.* 5, EICS (2021), 1–31. https://doi.org/10.1145/3457143

[8] Yan Chen, Lixian Huang, Lulu Li, Qi Luo, Ying Wang, and Jing Xu. 2007. The Experimental Approaches of Assessing the Consistency of User Interface. In *HCI. Interaction Design and Usability*. 420–427.

[9] Tim Clerckx, Kris Luyten, and Karin Coninx. 2004. The Mapping Problem Back and Forth: Customizing Dynamic Models While Preserving Consistency. In *Proc. of TAMODIA '04*. 33–42. https://doi.org/10.1145/1045446.1045455

[10] 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

[11] Elizabeth Furtado, Vasco Furtado, Wilker Bezerra Silva, Daniel William Tavares Rodrigues, Leandro da Silva Taddeo, Quentin Limbourg, and Jean Vanderdonckt. [n.d.]. An Ontology-Based Method for Designing Multiple User Interfaces. In *Proceedings of Int. Workshop on Multiple User Interfaces (MUI' 01)*. https://www.researchgate.net/publication/2567741_An_Ontology-Based_Method_for_Universal_Design_of_User_Interfaces

[12] Juan Manuel González-Calleros, Josefina Guerrero García, and Jean Vanderdonckt. 2013. Advance human-machine interface automatic evaluation. *Univers. Access Inf. Soc.* 12, 4 (2013), 387–401. https://doi.org/10.1007/s10209-013-0310-7

[13] Jonathan Grudin. 1989. The Case against User Interface Consistency. *Commun. ACM* 32, 10 (oct 1989), 1164–1173. https://doi.org/10.1145/67933.67934

[14] Wendy A. Kellogg. 1987. Conceptual consistency in the user interface: effects on user performance. In *Proc. of INTERACT '87*. North-Holland, Amsterdam, 389–394. https://doi.org/10.1016/B978-0-444-70304-0.50068-6

[15] Haibo Li and Dechen Zhan. 2006. Consistency of User Interface Based on Petri-Net. In *Proceedings of the 2006 International Conference on Advanced Web and Network Technologies, and Applications* (Harbin, China) *(APWeb'06)*. Springer, Berlin, Heidelberg, 846–852. https://doi.org/10.1007/11610496_116

[16] Wenqi Li, Chenyu Li, and Yuwei Yang. 2019. How Consistent Is Your GUI Design?. In *Conference Companion CSCW '19*. 278–282. https://doi.org/10.1145/3311957.3359449

[17] Rohit Mahajan and Ben Shneiderman. 1997. Visual and Textual Consistency Checking Tools for Graphical User Interfaces. *IEEE Trans. Softw. Eng.* 23, 11 (nov 1997), 722–735. https://doi.org/10.1109/32.637386

[18] Jeremy Mendel and Richard Pak. 2009. The Effect of Interface Consistency and Cognitive Load on User Performance in an Information Search Task. *Proc. of HFES '09* 53, 22 (2009), 1684–1688. https://doi.org/10.1177/154193120905302206

[19] Jan Meskens, Kris Luyten, and Karin Coninx. 2010. Jelly: A Multi-Device Design Environment for Managing Consistency across Devices. In *Proc. of AVI '10*. 289–296. https://doi.org/10.1145/1842993.1843044

[20] David Chek Ling Ngo, Lian Seng Teo, and John G. Byrne. 2003. Modelling Interface Aesthetics. *Inf. Sci.* 152, 1 (June 2003), 25–46. https://doi.org/10.1016/S0020-0255(02)00404-8

[21] Jeffrey Nichols, Brad A. Myers, and Brandon Rothrock. 2006. UNIFORM: Automatically Generating Consistent Remote Control User Interfaces. In *Proc. of CHI '06*. 611–620. https://doi.org/10.1145/1124772.1124865

[22] J. Nielsen. 1989. Coordinating User Interfaces for Consistency. *SIGCHI Bull.* 20, 3 (jan 1989), 63–65. https://doi.org/10.1145/67900.67910

[23] A. Ant Ozok and Gavriel Salvendy. 2000. Measuring consistency of web page design and its effects on performance and satisfaction. *Ergonomics* 43, 4 (2000), 443–460. https://doi.org/10.1080/001401300184332 PMID: 10801079.

[24] Thiago Rocha Silva, Marco Winckler, and Hallvard Trætteberg. 2020. Ensuring the Consistency between User Requirements and Task Models: A Behavior-Based Automated Approach. *PACMCI* 4, EICS (2020).

[25] John W. Satzinger. 1991. *User Interface Consistency across End-User Application Programs: Effect on Learning and Satisfaction.* Ph.D. Dissertation. Claremont Graduate School, Claremont, CA 91711, United States. UMI Order No. GAX91-23660e.

[26] John W. Satzinger and Lorne Olfman. 1998. User Interface Consistency across End-User Applications: The Effects on Mental Models. *Journal of Management Information Systems* 14, 4 (mar 1998), 167–193.

[27] Ben Shneiderman, Catherine Plaisant, Maxine S. Cohen, Steven Jacobs, and Niklas Elmqvist. 2016. *Designing the User Interface - Strategies for Effective Human-Computer Interaction, 6th Edition.* Pearson.

[28] Thiago Rocha Silva and Marco Winckler. 2017. A Scenario-Based Approach for Checking Consistency in User Interface Design Artifacts. In *Proc. of IHC '17*. https://doi.org/10.1145/3160504.3160506

[29] Sidney L. Smith and Jane N. Mosier. 1986. *Guidelines for Designing User Interface Software.* Technical report ESD-TR-86-278. The MITRE Corporation, Bedford, Massachusetts, USA. https://hcibib.org/sam/

[30] Leslie Tudor and Cheryl L. Coyle. 2011. Defining a Process for Cross-Product User Interface Consistency. In *HCI International 2011 – Posters' Extended Abstracts*. Springer Berlin Heidelberg, Berlin, Heidelberg, 91–95.

[31] Jean Vanderdonckt. 1995. Accessing guidelines information with Sierra. In *Proceedings of IFIP TC13 International Conference on Human-Computer Interaction, INTERACT '95, 27-29 June 1995, Lillehammer, Norway (IFIP Conference Proceedings)*, Knut Nordby, Per H. Helmersen, David J. Gilmore, and Svein A. Arnesen (Eds.). Chapman & Hall, 311–316.

[32] Jean Vanderdonckt and Xavier Gillo. 1994. Visual Techniques for Traditional and Multimedia Layouts. In *Proceedings of the ACM Int. Conf. on Advanced Visual Interfaces* (Bari, Italy) *(AVI 1994)*, Maria Francesca Costabile, Tiziana Catarci, Stefano Levialdi, and Giuseppe Santucci (Eds.). ACM, 95–104. https://doi.org/10.1145/192309.192334

[33] Seok Wangmi. 2015. A Framework Proposal of UX Evaluation of the Contents Coherency on Multi-screens. In *Proceedings of 4th International Congress on Advanced Applied Informatics* (Okayama, Japan) *(IIAI '15)*. IEEE Computer Society Press, 639–645. https://doi.org/10.1109/IIAI-AAI.2015.231

[34] Mathieu Zen and Jean Vanderdonckt. 2014. Towards an evaluation of graphical user interfaces aesthetics based on metrics. In *Proc. of RCIS '14*. IEEE, 1–12. https://doi.org/10.1109/RCIS.2014.6861050

[35] Michael Zuschlag. 2010. *Achieving and Balancing Consistency in User Interface Design.* https://www.uxmatters.com/mt/archives/2010/07/achieving-and-balancing-consistency-in-user-interface-design.php

A APPENDIX: UML ACTIVITY DIAGRAMS

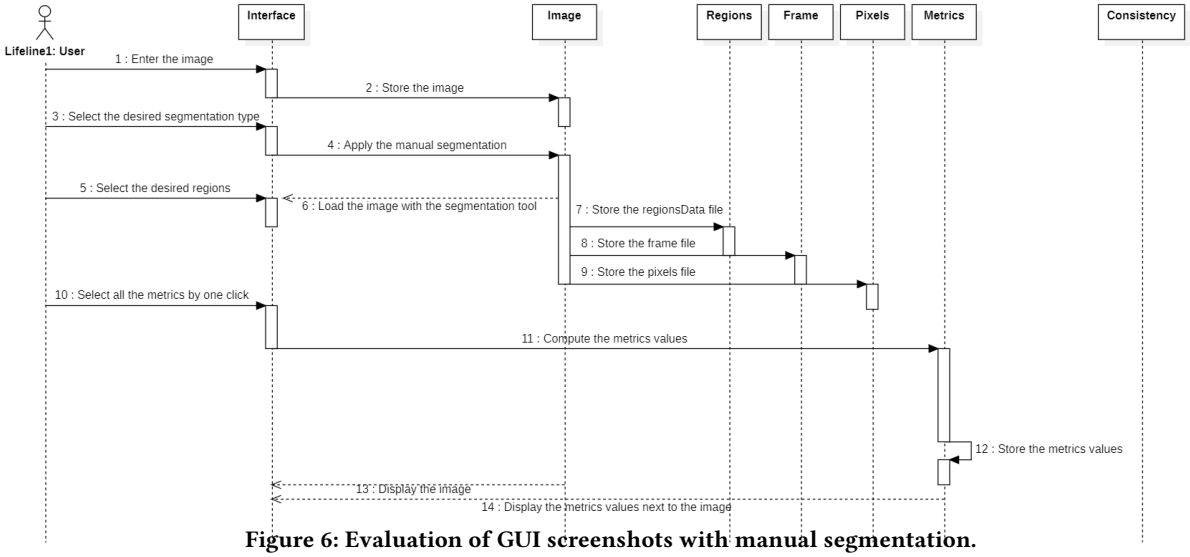


Figure 6: Evaluation of GUI screenshots with manual segmentation.

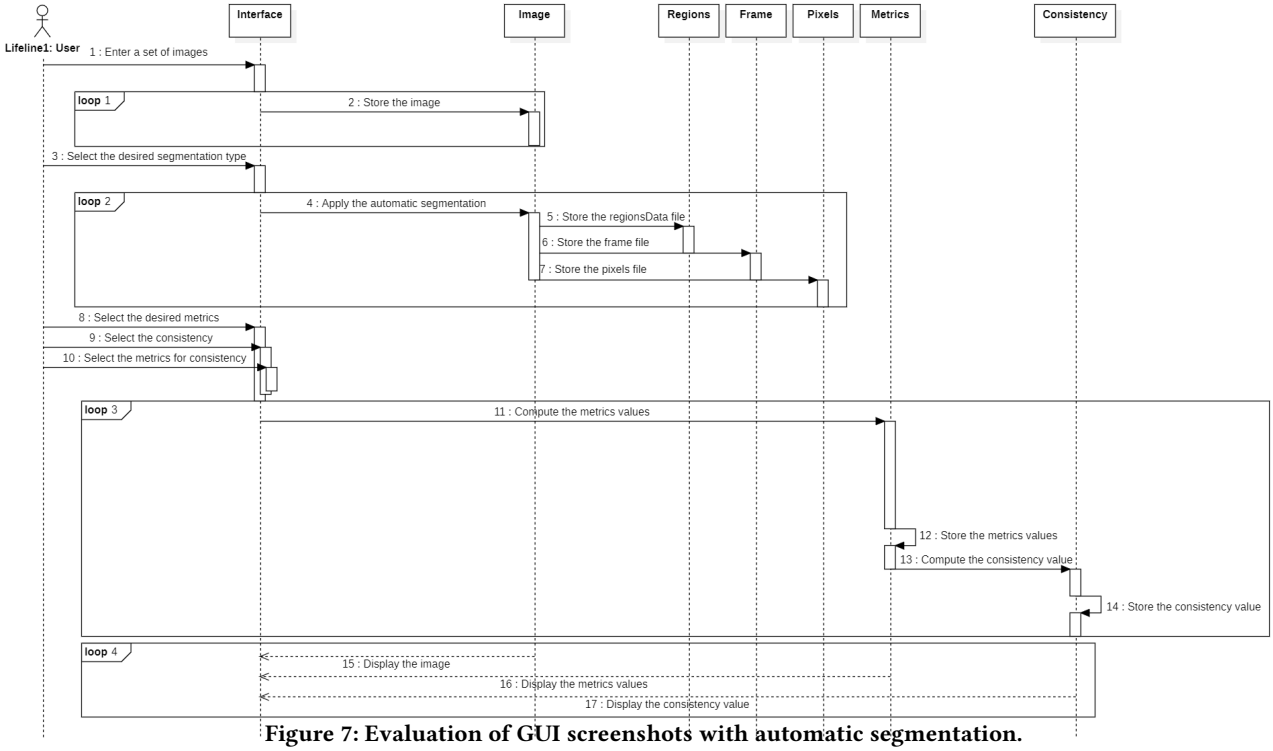


Figure 7: Evaluation of GUI screenshots with automatic segmentation.

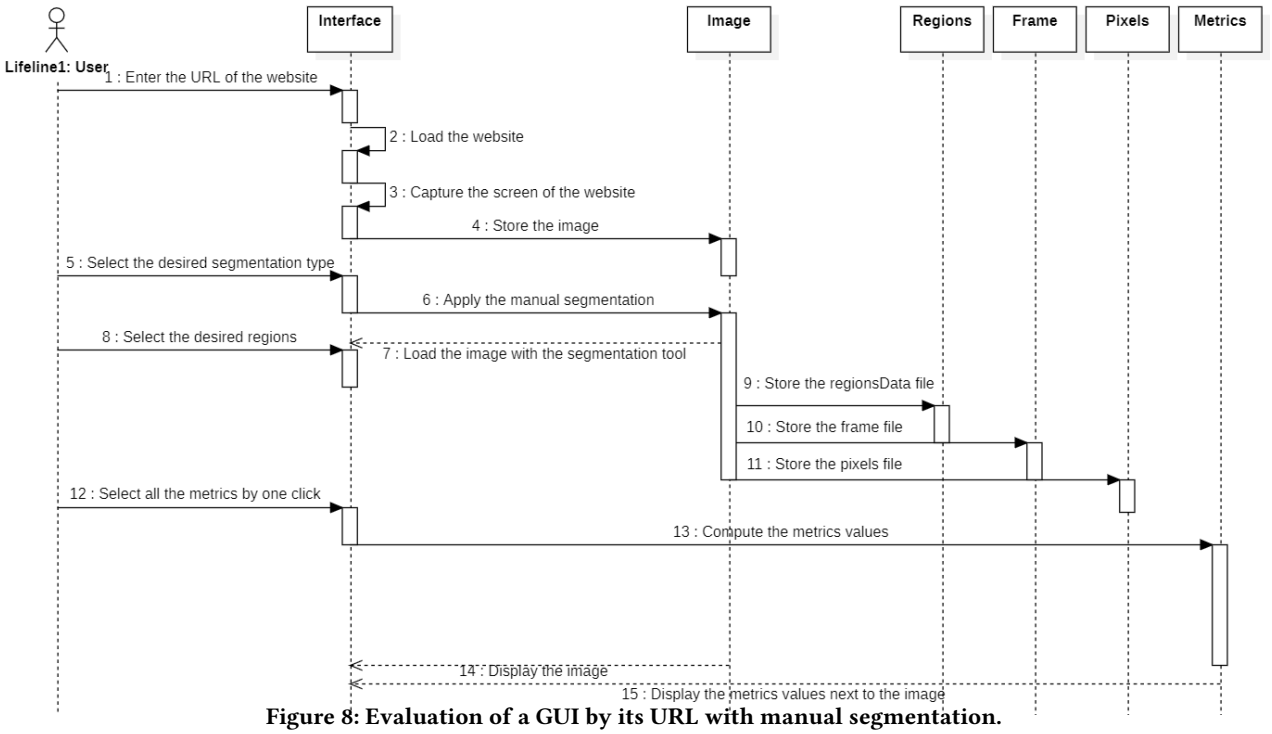


Figure 8: Evaluation of a GUI by its URL with manual segmentation.

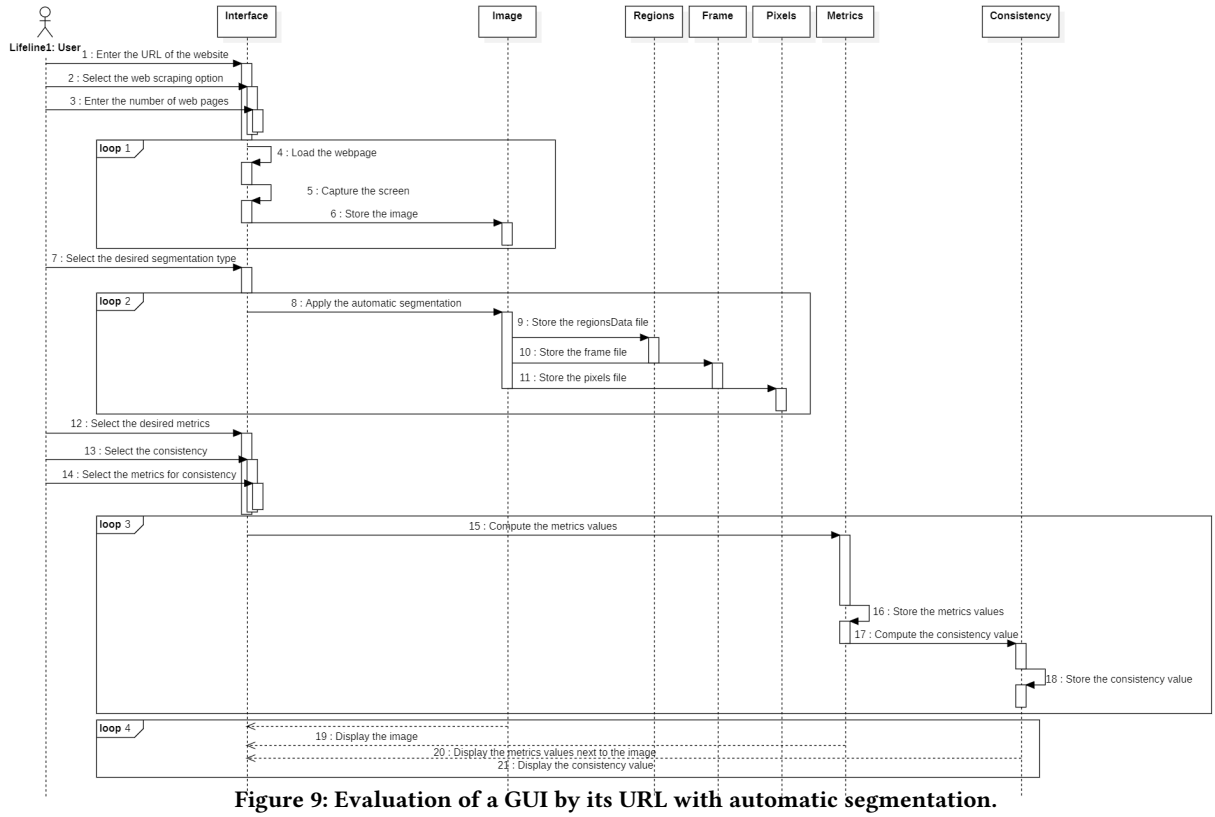


Figure 9: Evaluation of a GUI by its URL with automatic segmentation.