Matching User Interfaces to Assess Simple Web Applications

Marco Primo
Faculty of Sciences, University of Porto, Portugal

José Paulo Leal[1]
Faculty of Sciences, University of Porto, Portugal
CRACS – INESC, Portugal

Abstract

This paper presents ongoing research aiming at the automatic assessment of simple web applications, like those used in introductory web technologies courses. The distinctive feature of the proposed approach is a web interface matching procedure. This matching procedure verifies if the web interface being assessed corresponds to that of a reference application; otherwise, provides detailed feedback on the detected differences. Since web interfaces are event-driven, this matching is instrumental to assess the functionality. After mapping web interface elements from two applications, these can be targeted with events and property changes can be compared. This paper details the proposed matching algorithm and the current state of its implementation. It also discusses future work to embed this approach in a web environment for solving web application exercises with automatic assessment.

2012 ACM Subject Classification Information systems → Web interfaces; Applied computing → Computer-managed instruction

Keywords and phrases automatic assessment, web interfaces, learning environments

Digital Object Identifier 10.4230/OASIcs.ICPEC.2021.7

Category Short Paper

Funding *José Paulo Leal*: This work is financed by National Funds through the Portuguese funding agency, FCT – Fundação para a Ciência e a Tecnologia, within project UIDB/50014/2020.

1 Introduction

Automatic assessment of programming exercises is nowadays a standard practice in introductory programming courses [2]. It usually boils down to a black-box approach, where the assessed program is fed with test data, and the resulting output is compared with the output produced by a reference solution, for the same data. This approach is programming language agnostic and can be easily adapted to a wide range of settings. However, one of its shortcomings is the inability to deal with graphical user interfaces, in particular with web interfaces.

Graphical user interfaces in general, and web interfaces in particular, rely on user interaction for input rather than on data streams. On these applications, data is received as events that are targeted to different elements, without a predefined order. Moreover, web interfaces rely on the articulation of more than a single language – namely HTML, CSS, and JavaScript. These two characteristics prevent the use of simple black-box techniques, typically used for automatic assessment in algorithms and data structure courses.

To overcome this problem we propose a different approach to assess exercises in introductory web technologies courses. These exercises usually involve coding a simple web interface with a small number of widgets, such as buttons, editing fields, and selectors. Although

[1] corresponding author

simple, these web interfaces can be implemented in different ways. For instance, a button may be coded as either a `<button>` or `<div>` HTML element. To position these elements, different CSS techniques can be used, such as floating elements or *flexbox*, to name a couple. Last but not least, the exact position and dimension of these elements are immaterial, at least from a pedagogical perspective, provided they preserve geometric relationships (e.g. left-right, center-aligned).

The core of the proposed approach is the matching between two web interfaces: that of the application being assessed and that of a reference solution. The matching procedure is flexible enough to recognize equivalent elements that may have different types, slightly different sizes, or be organized using different structures. On a successful matching, the result is a mapping between the elements of the two web interfaces. On an unsuccessful matching, the result is a set of differences between the two web interfaces that may be used to generate feedback for the student. In the latter case, the assessment is complete and the student must first fix the detected issues.

A mapping between the elements assessed and the reference web interfaces are the basis for testing the web interface features. This stage is based on a sort of unit testing, where data is injected into certain elements as events, and assertions are made on the changes of properties in other elements. The tests that pass on the reference application are then applied to the application under assessment, after replacing the elements using the mapping. At this stage, failed assertions are also reported to the student as feedback, until the exercise is completely solved.

The remainder of this paper is organized as follows. Section 2 presents a literature survey on approaches to compare user interfaces. Section 3 details the proposed algorithm for matching web interfaces based on simple examples, and provides a few implementation details. Finally, Section 4 concludes with future work regarding the implementation of a web environment for solving web application exercises, capable of automatic assessment and feedback.

2 State of the art

The automatic evaluation of programming exercises has become a common practice in introductory programming courses, due to the growing number of new students [9, 10], which renders the manual assessment of programming exercises infeasible or demanding a large number of resources [9].

The literature describes several automatic assessment environments [1], however, none of these tackles efficiently the evaluation of graphical interfaces for educational purposes [1]. These environments help students by promoting feedback that steeps the learning curve [5]. Although the feedback provided by automatic assessment systems is not fully effective as an isolated practice, it becomes an important tool when combined with the teacher's feedback and its use is nowadays a standard in programming courses [10]. Automatic assessments can be divided into two broad categories: static or dynamic [2, 11]. The former analyses the code itself while the latter executes it and analyses its side effects.

To compare the two web pages graphically, different approaches can be taken. One of these approaches is structural comparison, as described in the article [15], this strategy consists of comparing the trees resulting from the two HTML documents. Despite being a possibility, this strategy may not be a solution when it comes to building an automatic evaluation system for web interfaces, given the fact that trees can be the same graphically and structurally different.

Computer vision is another approach that can be used to perform tests on graphical interfaces [3], this approach is based on artificial intelligence models. A construction of an algorithm capable of classifying images and composed by several steps, the main ones being the definition of an adequate classification system, selection of training, pre-processing of images, extraction of characteristics, selection of appropriate approach classifications, post-processing classification and precision evaluation [6]. Due to the inherent complexity of this class of algorithms, the usage of this algorithm can be prohibitive even for comparing simple web interfaces.

To the best of the authors' knowledge, no previous attempts were made to match web interfaces to assess programming exercises. The systems described in the literature to visually compare web pages have very different goals – such as detecting phishing sites [8, 4, 12] –, and are more focused on spotting small differences than ignoring them.

3 Web interface matching

The proposed approach to web interface matching is independent of their internal structures. This objective is achieved by making use of the element's properties of each page and the spatial relationships between them. Moreover, it was designed to produce incremental feedback that may help students overcoming their difficulties.

As in program assessment in general, web interface assessment compares a program against a model. It is a dynamic assessment since it compares the side effects of both programs' executions; in this case, the web interfaces they produce. To access one against the other, these web interfaces need to be related. This relation is created by using a comparison algorithm between web interfaces.

The approach we used to create an algorithm to relate web interfaces consists of mapping among leaf elements. Leaf elements, also known as *widgets* in user interface frameworks, are the elements used as leaves in a web interface tree structure, namely the leaves *DOM* (Document Object Model) [14] object structure. Non-leaf elements are less relevant since different structures, used for controlling the position of leaf elements, may lead to graphically similar web interfaces. Moreover, leaf elements of different types may have a similar look and feel. For instance, a button may be implemented both as a `<button>` or `<div>` element.

The proposed algorithm for comparing web interfaces operates of sets of leaf elements in 4 consecutive phases:

Initial sets creation: Creates two sets of leaf elements from each web interface, including their properties. This data is obtained using the *JavaScript* binding of the *DOM* interface.

Attribute refinement: In both sets, for each element, the properties that are not relevant to the algorithm are removed. The resulting sets are lists of *JSON* [7] objects containing properties, preserving some of the original properties, and computing new ones. Currently, the preserved properties are `name`, `id`, `tag`, `internal text`, and `src`. Two of the original properties – `offSetTop` and `offSetLeft` – are used to compute new properties to capture the spatial relations among elements.

Element mapping: The result of this phase is a set of pairs, one from each leaf element set. The rules that pair elements ensure that the most similar are created first. This greedy approach is used to ensure that a good mapping between all elements is efficiently created.

Feedback generation: The set of pairs is the basis for feedback generation. It should be noted that this set can only be created if initial sets have the same cardinality. Otherwise, feedback must identify the elements from the tested web interface that are missing, or

that could not be mapped. Even with a complete set of pairs, some of them may be partial matches. That is, some of their relevant properties may be different. Using this data higher granularity feedback is generated.

The similarity between elements of a pair is computed as a score obtained by comparing their properties. Due to some properties being more decisive than others, this score is a weighted sum.

The comparison of properties governing the elements' positions and sizes is the most challenging part of the proposed approach. This is done by considering relative spatial relationships between the elements of the same web interface. These relations dictate in which geometric quadrant the element is concerning the other.

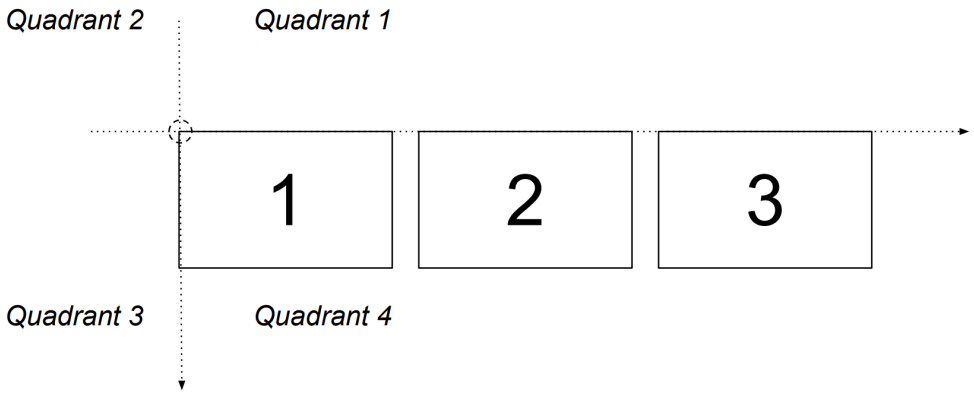

Figure 1 Example of quadrant based spatial relationships.

Consider a web interface with 3 non-overlapping leaf elements, aligned horizontally, with IDs 1, 2, and 3, as shown in Figure 1. Using the upper left corner as referential, following the convention in windowing systems, one can define relative spatial relations with the other elements. For element 1, elements 2 and 3 are in quadrant 4, as they are mostly to the right and below the upper left corner of element 1.

The attribute refinement phase adds new properties to each element counting the number of other elements (the element itself is not counted) lying in each of the 4 quadrants, using its upper left corner and referential. In the example of Figure 1: element 1 has 2 elements in the 4th quadrant; element 2 has 1 element in the 3rd quadrant and 1 in the 4th, and element 3 has 2 elements in the 3rd quadrant. The unmentioned quadrants of each element count 0 elements.

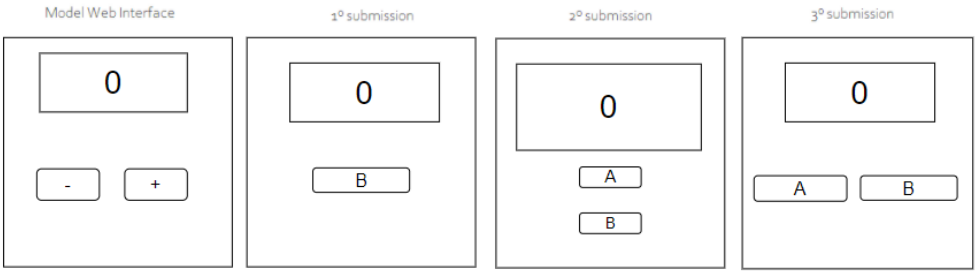

Figure 2 Example of incremental assessment and feedback.

Figure 2 illustrates successive steps in solving a web interface exercise made by the student until having it accepted, guided by the algorithm's feedback. The model application is a simple counter, with a label that displays the current count and two buttons, one to increase the count and another to decrease it. When the buttons are clicked, the displayed number is changed accordingly.

After the 1st submission, the algorithm cannot map all the leaf elements. However, it can identify a missing button and that information is reported as feedback to the student. Using that feedback, the student adds a button to the web interface. After the 2st submission, all elements of the student's web interface are mapped to those on the model web interface. However, they differ both on relative spatial relation and the buttons' internal text. To avoid overloading the student with excessive feedback [13] the information on non spatial errors is omitted from the feedback.

Subsequently, on a 3rd submission, all elements continue to be identified. However, the buttons' internal text still differs and that information is given as feedback. In the final submission, not shown in the figure, the student replaces "A" and "B", by "+" and "-", respectively, and the submission is accepted. There are still differences between the model and the submitted program but they are considered close enough for the intended purpose.

It should be noted that the algorithm can be configured to consider more (or less) properties as relevant. For instance, the size properties could have been considered relevant and the last submission would not be accepted, since the button's size is larger than those of the model web interface.

4 Future work

In our ongoing work, we plan to use the algorithm presented in Section 3 as the cornerstone of a web interface assessment environment. This environment will assess both the appearance and the interactivity of the web interfaces developed as exercises. The mapping algorithm maps elements in both web interfaces. Thus, functions that check invariants on the model interface, as a sort of unit tests, can be applied to the student's submission.

Consider again the model interface shown in Figure 2. A unit test can: 1) start by obtaining the label's initial value; 2) then send a click event to the increase button; 3) and, finally, check that label's value is incremented. The same unit test can be applied to a similar web interface, provided that the element references are replaced by the corresponding elements on the web interface being tested.

As the mapping between the web interfaces is done relatively, this means that measures and absolute positions become irrelevant, a web interface can be assessed as correct if it maintains the expected relative positions among the elements. With an evaluation that does not force the tested web interface to follow the same structure of the model, we expect to increase the fairness of the assessment environment.

References

1 Dr SM Afroz, N Elezabeth Rani, and N Indira Priyadarshini. Web application–a study on comparing software testing tools. *International Journal of Computer Science and Telecommunications*, 2(3):1–6, 2011.

2 Kirsti M Ala-Mutka. A survey of automated assessment approaches for programming assignments. *Computer Science Education*, 15(2):83–102, 2005. doi:10.1080/08993400500150747.

3 Tsung-Hsiang Chang, Tom Yeh, and Robert C. Miller. Gui testing using computer vision. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '10, page 1535–1544, New York, NY, USA, 2010. Association for Computing Machinery. doi:10.1145/1753326.1753555.

4 Igino Corona, Battista Biggio, Matteo Contini, Luca Piras, Roberto Corda, Mauro Mereu, Guido Mureddu, Davide Ariu, and Fabio Roli. Deltaphish: Detecting phishing webpages in compromised websites. In Simon N. Foley, Dieter Gollmann, and Einar Snekkenes, editors, *Computer Security – ESORICS 2017*, pages 370–388, Cham, 2017. Springer International Publishing.

5 H. Fangohr, Neil S. O'Brien, A. Prabhakar, and Arti Kashyap. Teaching python programming with automatic assessment and feedback provision. *ArXiv*, abs/1509.03556, 2015.

6 D. Lu and Q. Weng. A survey of image classification methods and techniques for improving classification performance. *International Journal of Remote Sensing*, 28(5):823–870, 2007. doi:10.1080/01431160600746456.

7 Felipe Pezoa, Juan L. Reutter, Fernando Suarez, Martín Ugarte, and Domagoj Vrgoč. Foundations of json schema. In *Proceedings of the 25th International Conference on World Wide Web*, WWW '16, page 263–273, Republic and Canton of Geneva, CHE, 2016. International World Wide Web Conferences Steering Committee. doi:10.1145/2872427.2883029.

8 S. Roopak and Tony Thomas. A novel phishing page detection mechanism using html source code comparison and cosine similarity. In *2014 Fourth International Conference on Advances in Computing and Communications*, pages 167–170, 2014. doi:10.1109/ICACC.2014.47.

9 Riku Saikkonen, Lauri Malmi, and Ari Korhonen. Fully automatic assessment of programming exercises. *SIGCSE Bull.*, 33(3):133–136, 2001. doi:10.1145/507758.377666.

10 Zahid Ullah, Adidah Lajis, Mona Jamjoom, Abdulrahman Altalhi, Abdullah Al-Ghamdi, and Farrukh Saleem. The effect of automatic assessment on novice programming: Strengths and limitations of existing systems. *Computer Applications in Engineering Education*, 26, February 2018. doi:10.1002/cae.21974.

11 M. Varga and M. Kvassay. Unit testing in data structures graphical learning environment. In *2019 17th International Conference on Emerging eLearning Technologies and Applications (ICETA)*, pages 797–804, 2019. doi:10.1109/ICETA48886.2019.9040071.

12 Gaurav Varshney, Manoj Misra, and Pradeep K Atrey. A survey and classification of web phishing detection schemes. *Security and Communication Networks*, 9(18):6266–6284, 2016.

13 Eric M Wilcox, J William Atwood, Margaret M Burnett, Jonathan J Cadiz, and Curtis R Cook. Does continuous visual feedback aid debugging in direct-manipulation programming systems? In *Proceedings of the ACM SIGCHI Conference on Human factors in computing systems*, pages 258–265, 1997.

14 Lauren Wood, Arnaud Le Hors, Vidur Apparao, Steve Byrne, Mike Champion, Scott Isaacs, Ian Jacobs, Gavin Nicol, Jonathan Robie, Robert Sutor, et al. Document object model (dom) level 1 specification. *W3C recommendation*, 1, 1998.

15 Jiří Štěpánek and Monika Šimková. Comparing web pages in terms of inner structure. *Procedia - Social and Behavioral Sciences*, 83:458–462, 2013. 2nd World Conference on Educational Technology Research. doi:10.1016/j.sbspro.2013.06.090.