# Enhancing feedback to students in automated diagram assessment[*]

**Helder Correia, José Paulo Leal, and José Paiva**

**CRACS & INESC-Porto LA, Faculty of Sciences,**
**University of Porto, Portugal**
`zp@dcc.fc.up.pt` `up201108850@fc.up.pt` `up201200272@fc.up.pt`

## Abstract

Automated assessment is an essential part of eLearning. Although comparatively easy for multiple choice questions (MCQs), automated assessment is more challenging when exercises involve languages used in computer science. In this particular case, the assessment is more than just grading and must include feedback that leads to the improvement of the students' performance.

This paper presents ongoing work to develop Kora, an automated diagram assessment tool with enhanced feedback, targeted to the multiple diagrammatic languages used in computer science. Kora builds on the experience gained with previous research, namely: a diagram assessment tool to computes differences between graphs; an IDE inspired web learning environment for computer science languages; and an extensible web diagram editor.

Kora has several features to enhance feedback: it distinguishes syntactic and semantic errors, providing specialized feedback in each case; it provides progressive feedback disclosure, controlling the quality and quantity shown to each student after a submission; when possible, it integrates feedback within the diagram editor showing actual nodes and edges on the editor itself.

**1998 ACM Subject Classification** D. Software; D.2 SOFTWARE ENGINEERING; D.2.6 Programming Environments; Interactive environments

**Keywords and phrases** automated assessment, diagram assessment, feedback generation,language environments, e-learning

**Digital Object Identifier** 10.4230/OASIcs.xxx.yyy.p

## 1 Introduction

Automated assessment is essential for effective eLearning. Both in formative and summative assessment, eLearning students need to have their exercises compared with standard solutions, so that they know if they are achieving the expected result. Any form of assessment, even if it is just a grade, is already *feedback* to the student. However, feedback should be more than just a distance to the correct solution. Students need to be guided, see evidence of their mistakes and receive suggestions to improve their performance [8].

When the set of all possible answers to an exercise is small, grading and feedback are fairly easy to automate. This is arguably the reason why multiple choice questions (MCQs) are so popular on eLearning. In fact, simple skills and superficial knowledge can be completely assessed using MCQs, but for some cases they are insufficient. For instance, it is impossible to assess a student proficiency on a language using only MCQs. This is obviously true in

[*] This work is financed by the ERDF – European Regional Development Fund through the Operational Programme for Competitiveness and Internationalisation – COMPETE 2020 Programme, and by National Funds through the FCT – Fundação para a Ciência e a Tecnologia (Portuguese Foundation for Science and Technology) within project POCI-01-0145-FEDER-006961..

natural languages, such as English and Portuguese, and is also the case for the artificial languages used in computer science. This fact lead to the development of several systems for assessing both programming languages [4] and diagrammatic languages [11, 2]. Nevertheless, feedback in these systems is still largely an open issue [5] .

Diagram assessment has been less researched than program assessment, which is understandable since programs are more relevant than diagrams in computer science. However, programs are much more difficult to assess than diagrams, since their semantics is more complex. That is, a program has an operational semantics that needs to be checked with test data but diagrams have only a declarative semantics. Feedback on a diagram exercise can be solely based on the differences between the student's diagram and a solution. The relevance of the research on diagram assessment feedback is twofold: diagram languages are studied in several computer science disciplines, such as theory of computation (DFA), databases (EER) and software modeling (UML), thus it is useful for teaching those subjects; the tools and techniques developed for diagrammatic languages may later on be extended to more complex languages such as programming languages.

The research presented in this paper builds on previous work to develop the components for diagram assessment, namely a computer language learning environment [9], a diagram editor [6] and a a graph comparator [12]. Diagrams are modeled by graphs, hence it is possible to compare two diagrams by computing the differences between their model graphs. For large graphs, the computational complexity is prohibitive, but using heuristics and for graphs of the size typically used in programming exercises, this method is effective. However, the validation of the graph comparator revealed several issues related to feedback generation.

This paper presents Kora, a component for assessing diagrams with enhanced feedback. It was designed to support any diagrammatic language used in computer science. Nevertheless, some of its features were inspired by existing UML editors and it will validated with class and use case diagrams. Hence, Section 2 surveys in particular the existing editors and assessment systems for that language. Kora relies on components resulting from previous research that are described in Section 3. On on the contribution of this work is the Diagrammatic Language Definition Language (DL2) that is presented in section 4. The enhanced feedback features provided by Kora required a redesign of diagram editor Eshu, as explained in Section 5. The design and implementation of the Kora component is presented in Section 6. The final section addresses the work currently being done and the planned validation of Kora.

## 2 Related work

Kora supports the creation and assessment of diagram exercises of any type, with visual and textual feedback. To the best of authors' knowledge, no other tool described in the literature includes all these features. Hence, this section reviews several works including some of these features.

Most of the existent automatic diagram assessment systems are designed for a specific diagram type. Some examples of these system are deterministic finite automata (DFA) [2, 10], UML class diagrams [1, 11, 15], Entity-Relationship diagrams [3], among others.

There are many diagram editing tools targeted to UML and most of them are commercial. Because they are developed for companies that normally use them for the modeling of complex systems, they present more features and functionalities (e.g forward engineering,reverse engineering). Most of these tools are visual tools defined for multi-domain modeling (e.g computational modeling) that support UML diagram modeling or drawing. Examples of this kind of tools are *MagicDraw*[1] and *Modelio*[2].

Nevertheless, there are a few non-commercial UML tools such as *ArgoUML* [14], and *Dia*[3]. These are usually developed by research groups with pedagogical scope, they usually have fewer features and functionality than commercial tools. However, in general, they are tools developed for the domain of modeling UML diagrams and present models that faithfully follow the specification UML.

A growing number of UML editing tools are deployed on the web, such as *Cacoo*[4], These tools typically allow real-time multi-user collaboration in diagram editing, with specific features that facilitate this mode of editing (chat and version control). They are tools for drawing and not modeling, they have few features and functionalities and it is mandatory to have an account.

From the diagram assessment viewpoint, critiquing systems are a relevant feature of many UML editing and modeling tools. A critiquing system acts on modeling tools to provide corrections and suggestions on the models to be designed. Much research has been devoted to critiquing tools and they are incorporated in systems such as *ArgoUML* [14], *ArchStudio5* [5] *ABCDE-Critic* [13].

## 3 Background

In project Eshu [6], we development an extensible diagram editor, embeddable in Web applications that require diagram interaction, such as modeling tools or e-learning environments. Eshu is a JavaScript library with an API that supports its integration with other components, including importing/exporting diagrams in JSON.In order to validate the API of Eshu we created an EER diagram editor in *Javascript* using the library provided by Eshu and HTML5 canvas. The editor allows to edit ERR diagram, import / export diagram into JSON format, apply ERR language restrictions in diagram editor (constraints on links), display visual feedback on EER diagram submissions. The editor has been integrated into the Enki [9] with a diagram evaluator and was used in database course to edit and evaluate EER diagrams.

Diagrams are schematic representations of information that, ignoring the positioning of its elements, can be abstracted in graphs. Based on this, structure driven approach to assess graph-based exercises was proposed [12]. Given two graphs, a solution and an attempt of a student, this approach computes a mapping between the node sets of both graphs that maximizes the students grade, as well as a description of the differences between the two graph. Uses an algorithm with heuristics to test the most promising mappings first and prune the remaining when it is sure that a better mapping cannot be computed.

Enki [9] is a web-based IDE for learning programming languages, which blends assessment (exercises) and learning (multimedia and textual resources). It integrates with external services to provide gamification features and to sequence educational resources at different rhythms according to students' capabilities. The assessment of exercises is provided by the new version of Mooshak [7] – Mooshak 2.0 –, which among other features, allows the creation of special evaluators for different types of exercises.

[1] https://www.nomagic.com/products/magicdraw.html
[2] https://www.altova.com/umodel.html
[3] http://dia-installer.de/
[4] https://cacoo.com
[5] https://basicarchstudiomanual.wordpress.com/

## 4 Language configuration-DL2

Kora was designed to be extensible, to be able to incorporate new diagrammatic languages defined by an XML configuration file. This file includes configurations for syntactic feedback and editor. It configures types of nodes, types of edges, restrictions of the language, among others, that are used while validating the syntax. It includes configurations of the editor and toolbar style that applied on Eshu.

The configuration file consists of two top elements `Style` and `Diagram`. Type `Style` contains information such height, width, background and grid of the editor and the toolbar. Type `Diagram` configures the syntax of the language (nodes, edges and constraints), and has two types of elements, `nodeTypes` and `edgeTypes`, and two attribute, `name` and `pathFile`. The attribute `name` contains the name of the language, `pathFile` contains the path of the configuration file. Elements of *nodeTypes* contain a set of `nodeInfo` and each `nodeInfo` is a configuration of a node (type, svg image, label, URL for node type information, visible properties in the configuration window, type of connections of the node and degree in/out of this node). Elements of `edgeTypes`, similar to `nodeTypes`, contain a set of `edgeInfo`, and each `edgeInfo` contains the configuration of an edge type.

## 5 Eshu 2.0

A diagram is composed of a set of Node and a set Edge; Nodes have a position and dimension; Edges connect a source and a target node. Although Eshu 2.0, similarly to Eshu 1.0 [6], follows an object-oriented approach for *Javascript*, it separates the data part from the visualization and editing part.

Eshu 2.0 consists of three packages: `eshu`, `graph` and `commands`. The package `graph` has the classes responsible for creating nodes and edges, storing the graph (`Quadtree`) and operating on the data of the graph (insert, remove, save changes and select an element). Package `eshu` contains the classes responsible for the user interface, including handlers for user interaction, methods to export and import the graph of the diagram in JSON format, methods to present visual feedback in the diagram editor, among many others. The package `commands` contains the classes that are responsible for the implementation of operations, such as undo, redo, paste, remove or resize.

One of the main improvements of Eshu 2.0 is the extensibility of nodes and edges. In Eshu 1.0, the creation of a new type of node (or edge) involves the creation of a new class that extends `Vertice` (or `Edge` for edges) and defines the method `draw`. With Eshu 2.0, a new type of node (or edge) can be inserted by only adding a `nodeConfig` (or `edgeConfig`) element to `nodeTypes` (or `edgeTypes`), in the configuration file. This element contains general information for a node (or edge), such as its SVG image path (used to represent it in the UI), type name, constraints on connections, among others.

Eshu is a pure JavaScript library, hence it can be integrated in most web applications. However, some frameworks, such as Google Web Toolkit (GWT), use different languages to code the web interfaces, in this case Java. To enable the integration of Eshu in GWT applications, a binding to this framework was also developed. The binding is composed of a Java class (that is converted to JavaScript by GWT) with methods to use the API, implemented using the JavaScript Native Interface (JSNI) of GWT.

The undo and redo commands are very important to the user while editing the graph. These two operation were not included in the first version of Eshu [6], but were now added. To facilitate the integration of these operations, a set of classes that implement the command

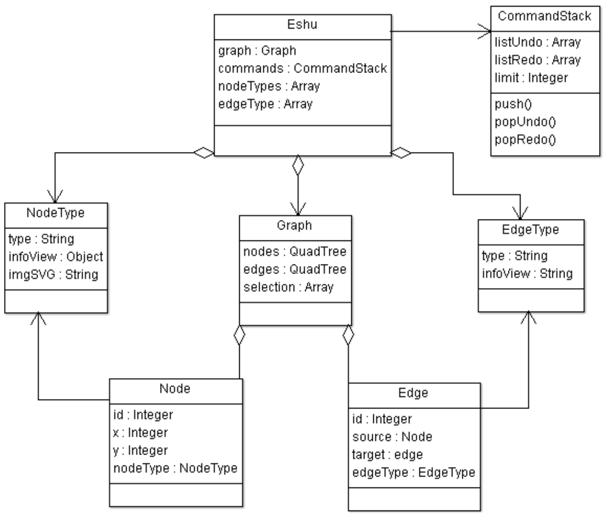

**Figure 1** Diagram Class Eshu

design pattern were developed. Now, operations, such as insert, delete and paste, are encapsulated as an object allowing to register them in a stack, and thus pop or push them.

Also, the API allows the host application to send feedback in the form of changes to the existing diagram. If these changes are deletions or modifications, they can be rendered by displaying the existing nodes and edges with a different color (blue – insert, red – delete). However, if the difference is a node insertion then it has to be positioned by Eshu. The layout of these new nodes is computed using a force-directed algorithm. In this approach, nodes repel each other according to Coulomb's law, as if they were electrically charged particles with the same signal, and edges bind them together as springs following Hooke's law.

## 6 Kora component

The Kora component is divided into two parts, `client` and `server`. The `client` part is integrated on the web interface and is responsible for running the Eshu editor, as well as handling user actions and presenting the feedback. The `server` part is responsible for evaluating diagrams, generating feedback, and exchanging information with the client side, such as language configurations.

A diagram is a schematic representation of information. This representation has associated to itself elements that have certain characteristics and a positioning in the space. By abstracting the layout (the position of the elements), the diagrams can be represented as graphs. The approach that is intended to follow for the assessment of the diagrams is the comparison of the graphs. Thus, it is possible to analyze the contents of the diagram without giving relevance to its positioning or graphic formatting.

In Eshu 1.0, types of connections are checked during creation editing, that is, if a source and target nodes could not be connect it would be reported immediately. However, during the validation of Eshu 1.0, it was noticed that the editor was getting slower as the number of nodes increased, although not all syntactic issues were actually covered. Also, syntactically incorrect graphs were causing problems in the generation of feedback by the evaluator. Due to these issues, syntactic verification was moved to Kora.

The diagram assessment in the system is split in two parts: syntactic assessment and semantic assessment. The syntactic assessment involves the conversion of the JSON file to a graph structure, and validation of the language syntax. It consists of validating the structural organization of the language, based on the set of rules, defined in the configuration file, for the types of nodes and edges. In this phase, the following tasks are done: validation of the types for the language; validation of the edges – for each edge it is checked if the type, source and target are valid; validations of the nodes – the degree of in and out are valid; validation of the number of connected components in the graph. The semantic assessment has to do with the comparison of the diagrams and follows graph assessment algorithm [12]. The evaluator receives a graph as an attempt to solve a problem and compares it with a graph solution, aims to find out which mapping of the solution nodes in nodes attempt to minimize the set of differences and therefore maximize the classification. For this, it is necessary to find out which solution node corresponds to the attempt node.

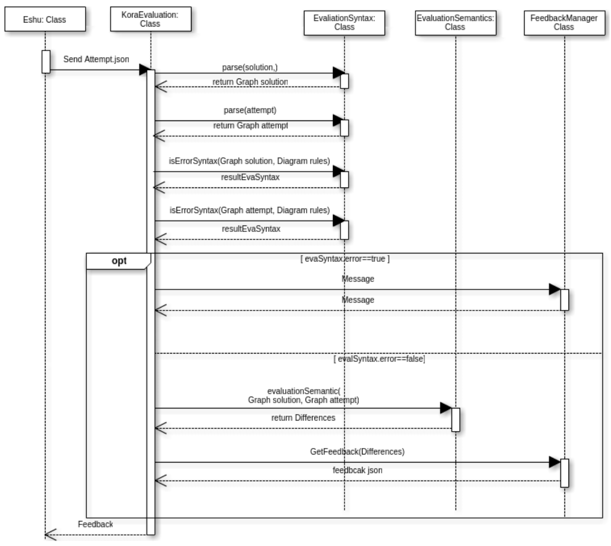

**Figure 2** Assessment diagram in Kora system

Figure 2 presents the UML sequence diagram of the diagram assessment in the Kora system. The Kora client gets the graph of the diagram in JSON format through a function of the Eshu API – `Eshu exportGraph()`. It parses the JSON graphs of the solution and the attempt, gathering the information necessary, in the form of graph, to represent them in the next assessments of these diagrams. Then, the *Kora* system performs syntactic validation and reports the existence of any syntax error, aborting the evaluation if a syntax error exists. If it does not contain any syntax error, it proceeds to semantic evaluation. This evaluator receives two graphs, the attempted graph and a solution graph. In the wrong answers, the errors are located and inserted into the lists of differences. Based on this list, the respective feedback is generated and a classification is calculated, so that the diagram can be improved.

The semantic assessment provided by Kora is based on the differences computed by a graph evaluator. The graph evaluator compares two graphs (attempt and solution), returns a set of differences and based on these differences is generated a set of feedback that is presented in Eshu, both in visual and textual form. However, when the student's attempt is far from the solution, it reports too many differences.

To cope with this problem Kore uses an incremental feedback generator. The generator uses several strategies to summarize a list of differences in a single message. The most general message that was not yet presented to the user is then selected as feedback.

Kora uses a repertoire of strategies to summarize a list of differences. Some strategies manage to condense several differences. For instance, several differences reporting a missing node of the same type may be condensed in the message "$n$ missing nodes of type $T$". Another strategy may select one of these nodes and show its label. An even more detailed strategy may show the the actual missing node on the diagram. A particular strategy may not be applicable to some list of differences. In this case no message is produced.

The resulting collection of feedback messages is sorted according to generality. General messages have precedence over specific messages. However, if a message was already provided as feedback than it is not repeated. The following message is reported instead. Using this approach, messages of increasing detail are provided to the student if she or he persist on the same exact error.

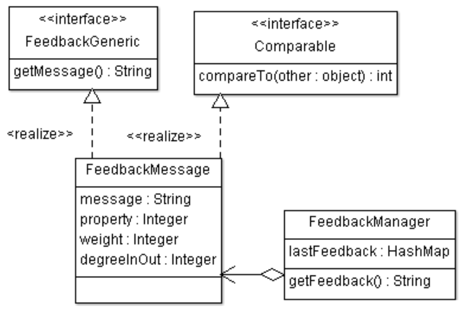

**Figure 3** Feedback Manager

Figure 3 presents the UML class diagram of the feedback implementation. The class `FeedbackMessage` contains the feedback information, including message, property number, weight, and in / out degrees (if it is a node). The property number indicates the property to which the message refers, the weight defines how much important is the mistake of the student, the degree of input/output allows to determine the importance of the node comparing to other nodes (i.e. higher degree, generally, means higher importance), and the message is the message itself. The class `FeedbackManager` generates and selects the feedback to be sent to the student. From the list of differences that is returned by the graph evaluator, it is generated a list of `FeedbackMessage`. From this list, the feedback already sent to the student is removed, and the remaining is sorted based on the fields of the `FeedbackMessage` class. The first `FeedbackMessage` from the list is selected and sent to the student.

## 7 Current and Future Work

Kora is work in progress. The project is in the final stage of development, just before validation. The design of Kora, including the diagrammatic language definition language, is already concluded. The implementation of the components described in this paper is in an advanced stage. Currently, most of the development effort is in the integration of Kora in Eshu, the learning environment where it will be deployed. In parallel, the definitions for two types of UML diagrams, namely class and use case, are also in development. Exercises for these diagram types will be used in the validation of Kora.

The research question that Kora aims to answer is: can feedback be enhanced by processing the output of an evaluator? Thus, the validation of the proposed approach will compare the efficacy of feedback with and without Kora. An experiment on the effect of Kora as a "treatment" to improve the efficacy of feedback is also being designed. Two groups of randomly chosen students will solve the same exercises, one group will receive raw feedback (just a grade and a list of differences) and the other will receive feedback processed by Kora. Both quantitative and qualitative differences are expected in the outcomes of the two groups. A number of variables will be measured to quantify those differences, including: the percentage of solved exercises, the number of submissions per problem and the time spent per exercise. To estimate the qualitative difference, students of both groups will be asked to fill in a questionnaire on their experience using diagram assessment tool.

## References

**1** Noraida Haji Ali, Zarina Shukur, and Sufian Idris. A design of an assessment system for uml class diagram. In *Computational Science and its Applications, 2007. ICCSA 2007. International Conference on*, pages 539–546. IEEE, 2007.

**2** Rajeev Alur, Loris D'Antoni, Sumit Gulwani, Dileep Kini, and Mahesh Viswanathan. Automated grading of dfa constructions. In *IJCAI*, volume 13, pages 1976–1982, 2013.

**3** Firat Batmaz and Chris J Hinde. A diagram drawing tool for semi–automatic assessment of conceptual database diagrams. 2006.

**4** Christopher Douce, David Livingstone, and James Orwell. Automatic test-based assessment of programming: A review. *Journal on Educational Resources in Computing (JERIC)*, 5(3):4, 2005.

**5** Hieke Keuning, Johan Jeuring, and Bastiaan Heeren. Towards a systematic review of automated feedback generation for programming exercises. In *Proceedings of the 2016 ACM Conference on Innovation and Technology in Computer Science Education*, pages 41–46. ACM, 2016.

**6** José Paulo Leal, Helder Correia, and José Carlos Paiva. Eshu: An extensible web editor for diagrammatic languages. In *OASIcs-OpenAccess Series in Informatics*, volume 51. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2016.

**7** José Paulo Leal and Fernando Silva. Mooshak: A web-based multi-site programming contest system. *Software: Practice and Experience*, 33(6):567–581, 2003.

**8** Robin Mason and Frank Rennie. *Elearning: The key concepts.* Routledge, 2006.

**9** José Carlos Paiva, José Paulo Leal, and Ricardo Alexandre Queirós. Enki: A pedagogical services aggregator for learning programming languages. In *Proceedings of the 2016 ACM Conference on Innovation and Technology in Computer Science Education*, pages 332–337. ACM, 2016.

**10** Zarina Shukur and Nurul F Mohamed. The design of adat: A tool for assessing automata-based assignments. *Journal of Computer Science*, 4(5):415, 2008.

**11** Josep Soler, Imma Boada, Ferran Prados, Jordi Poch, and Ramon Fabregat. A web-based e-learning tool for uml class diagrams. In *Education Engineering (EDUCON), 2010 IEEE*, pages 973–979. IEEE, 2010.

**12** Rúben Sousa and José Paulo Leal. A structural approach to assess graph-based exercises. In *International Symposium on Languages, Applications and Technologies*, pages 182–193. Springer, 2015.

**13** Cleidson RB Souza, JS Ferreira, Kléder Miranda Gonçalves, and Jacques Wainer. A group critic system for object-oriented analysis and design. In *Automated Software Engineering, 2000. Proceedings ASE 2000. The Fifteenth IEEE International Conference on*, pages 313–316. IEEE, 2000.

**14** Tigris.org. Welcome to ArgoUML. `http://argouml.tigris.org/`. Accessed: 2017-04-05.

**15** Vinay Vachharajani and Jyoti Pareek. A proposed architecture for automated assessment of use case diagrams. *International Journal of Computer Applications*, 108(4), 2014.