Eshu: An Extensible Web Editor for Diagrammatic Languages*

José Paulo Leal[1], Helder Correia[2], and José Carlos Paiva[3]

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

2 **CRACS & INESC-Porto LA, Faculty of Sciences, University of Porto, Porto, Portugal**
up201108850@fc.up.pt

3 **CRACS & INESC-Porto LA, Faculty of Sciences, University of Porto, Porto, Portugal**
up201200272@fc.up.pt

Abstract

The corner stone of a language development environment is an editor. For programming languages, several code editors are readily available to be integrated in Web applications. However, only few editors exist for diagrammatic languages. Eshu is 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. Eshu was already integrated in a pedagogical environment with automated diagram assessment, configured for extended entity-relationship diagrams, that served as basis for an usability evaluation.

1998 ACM Subject Classification D.2.6 Programming Environments; Interactive environments

Keywords and phrases Diagram assessment, language environments, automated assessment, e-learning

Digital Object Identifier 10.4230/OASIcs.SLATE.2016.12

1 Introduction

Programming is about languages, but not all languages used in programming are textual. Visual languages are typically used when it is necessary to abstract complex concepts and highlight relationships among them. Visual languages replace named types by simple shapes, such as rectangles or ellipses, that our brain grasps immediately. They also replace identifiers used in textual representations by lines connecting occurrences of these concepts. Still, visual languages have drawbacks when compared to textual languages. The number of types that can be represented with simple shapes is limited and the same goes for the number of connections that a person perceives using lines.

* This work is partially financed by the ERDF – European Regional Development Fund through the Operational Programme for Competitiveness and Internationalisation – COMPETE 2020 Programme and by the FCT within project POCI-01-0145-FEDER-006961 and project "NORTE-01-0145-FEDER-000020" financed by the North Portugal Regional Operational Programme (NORTE 2020), under the PORTUGAL 2020 Partnership Agreement and through the European Regional Development Fund (ERDF).

Visual languages in programming are mainly diagrams used for modelling[1]. The most popular example is arguably the (Extended) Entity-Relationship diagrams (EER) used for modelling databases. Another example are Deterministic Finite Automaton (DFA) diagrams used for modelling simple computations. The Unified Modelling Language (UML) also specifies a number of diagram types, including the popular Class Diagram, that can be seen as an extension of EER, and the state transition diagram that is an extension of DFA.

A diagram editor is the core component of an environment for diagrammatic languages. The goal of Eshu – the library described in this paper – is to provide diagram editing to Web based applications. Typical clients of Eshu are e-learning systems for diagrammatic languages, in particular those with automated evaluation of diagrams, that provide feedback to the student and need it to be displayed as diagram fragments (inserted and removed nodes and edges) overlaying the original diagram.

There are several programs for editing diagrams, including e-learning systems for certain diagrammatic languages. A survey of these systems is presented in Section 2. This survey covers also libraries for displaying and editing diagrams on web applications, with features in common with Eshu. Nevertheless, they lack the concept of language, both for supporting guided edition and to exchange diagrams with other tools, such as evaluators.

The concept of language is essential in Eshu. It supports the basic concepts of diagrams, such as nodes and edges, and basic operations on them. Nodes and edges may have their own visual syntax according to their type. Restrictions on the type and number of edges connected to a type of node can also be defined. Diagrammatic languages aggregate a collection of types of edges and nodes. Eshu has its own JSON based format for exchanging diagrams of any kind. This format can also be used for transmitting differences that Eshu is able to display over the current diagram. The features of Eshu as well as its design and implementation are detailed in Section 3.

Eshu was integrated in Enki, a web environment for learning computer science languages. The integration is described in Section 4 and served as a validation of the proposed library. It validated both the interoperability features of Eshu, in particular with the graph evaluation module [13], and the usability of the user interface. Section 5 summarizes the main contributions of this work and discusses the opportunities for future research.

2 State of the Art

The most used diagram editors are rich client applications, such as Microsoft Visio or Dia. A few libraries embeddable in web applications, such as mxGraph or GoJS, are also available. This section surveys the features of the systems that served as inspiration for the design of Eshu.

Microsoft Visio[2] is an application for creating diagrams in Windows. The strengths of Visio are the professional and technical diagrams with vector images, which can be magnified and manipulated. Visio can be used to create diagrams of various types, such as organization charts, flowcharts, data modeling (using UML or other graphical notation), network diagrams, floor plans, posters, etc.. Besides the ability to export diagrams into a variety of different formats (for example: PDF, SVG, DWG), it also enables a publishing model in Web format and export/import in XML formats. There are only versions of Visio for Windows. Also, it is a commercial software and it lacks diagram validation.

[1] There are also visual programming languages, but these are seldom used in computer science.
[2] `https://msdn.microsoft.com/en-us/library/office/ff604964(v=office.14).aspx`

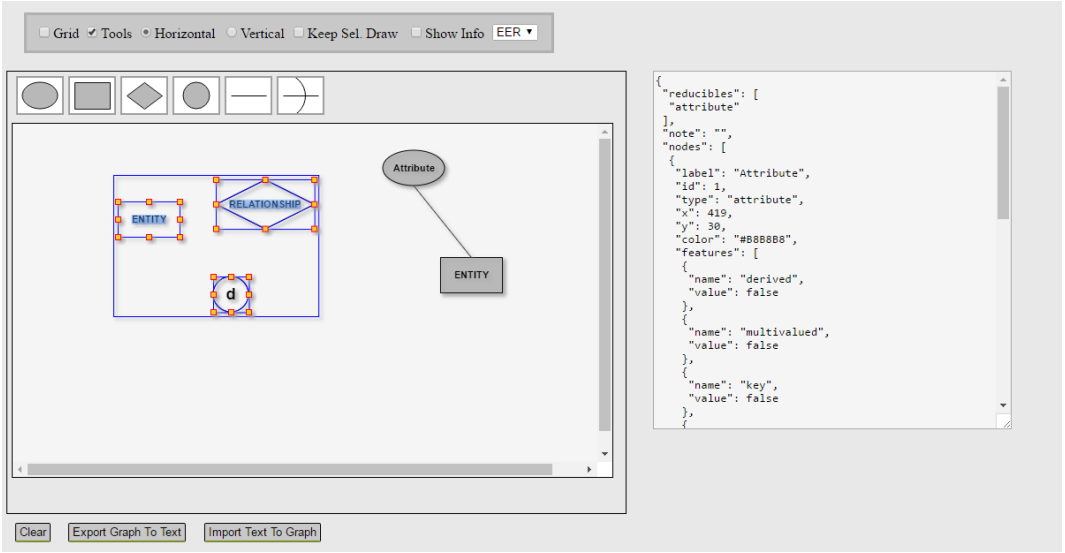

Figure 1 Eshu test application.

Inspired by the Windows program Visio, Dia[3] is a free diagram drawing software running on GNU/Linux, MacOS X, Unix, and Windows. Most diagram objects have handles to which lines can be connected to form graph structures. When the objects are moved or resized, the connections follow their respective objects. Dia has stencils for a wide variety of use cases such as user interface layout, organizational chart, entity relationship diagrams, UML diagrams, network diagrams and flowcharts. It can load and save diagrams to a custom XML format, save files in various formats such as EPS, SVG, xfig, WMF and PNG, among others. As a drawback for educational purposes, Dia allows nodes and connectors from various categories to be mixed together in the same diagram.

The product family library mxGraph [7] provides resources to applications that display interactive charts and graphs with implementations in various technologies. The JavaScript implementation supports several features such as file I/O using XML and JSON, dynamic styling of nodes and edges, event processing, folding of subtrees in acyclic graphs and node grouping.

GoJS [11] is a pure JavaScript library for the implementation of interactive diagrams in browsers. It offers advanced features for user interactivity, such as drag, copy and paste the text editing site, automatic layouts, data binding and templates, event handlers and select groups of nodes. Usually, it runs completely in a Web browser without any need for JavaScript libraries or frameworks, facilitating its integration. It supports diagram serializing in JSON.

Most of the available automatic diagram assessment systems were designed for a specific diagram type and use tailor made diagram editors . Examples of these single diagram types addressed by existing systems are deterministic finite automata (DFA) [2, 10], UML class diagrams [1, 12], UML use case diagrams [14], Entity-Relationship diagrams [3], among others.

[3] `https://wiki.gnome.org/Apps/Dia`

3 Design and implementation

Eshu is a diagram editor running on a web browser, written in JavaScript and using the HTML 5 canvas. This section details the main design points, starting with the user interface, covering also its interoperability and extensibility features, and ending with a few implementation details.

3.1 User interface

Eshu is not a complete application, it is a complex widget comparable to a rich text editor. Its user interface appears in the context of a host application. During the development of Eshu, a test application was developed to mock a host application. Figure 1 presents a screenshot of the test application. In fact, the user interface of Eshu is just the central rectangle displaying a diagram, including the toolbar for selecting nodes and edges. All the other widgets, such as check boxes, buttons and the text window, are part of the test application. These widgets interact with Eshu through its API, in a similar manner as a host application would do.

Eshu is started by the host application like any widget provided by a framework. When started, it has a certain width and height, that by default is also the size of the canvas where the user draws the diagram. If the user drags an object to the right or to the bottom then the canvas will automatically enlarge. The size of the Eshu widget will remain the same and scroll bars will be added to allow the navigation on the enlarged diagram.

The types of diagram elements, nodes and edges, available for the current diagrammatic language are displayed in the toolbar, right above the canvas. As mentioned previously, this panel is an integral part of Eshu. The toolbar may be positioned both vertically and horizontally, or even hidden, by using the appropriate calls to the API. After selecting an element type, the user may insert a new instance anywhere on the diagram and it may even overlap other elements.

Newly inserted nodes are selected, meaning that they will be the focus of the subsequent editing operations. If the user types, the label will be replaced, and if he drags its handles (the small squares on the borders), the node will be resized. Node size is also automatically adjusted according to the label text.

Eshu can also display feedback. Figure 2 presents an example of the feedback shown to the user. In the feedback, green nodes are modifications, blue nodes are additions and red nodes are deletions.

3.2 Interoperability

The main purpose of Eshu is to be integrated in web applications, hence interoperability is a particularly important facet of its design. Eshu has an extensive API that allows the host application to control it. Some of the most important methods of the API require a diagram serialization, to import or export them. Eshu uses its own JSON based serialization, specified using JSON-Schema [4].

Figure 3 presents, as UML classes, the data schema used by the API. As expected, a diagram is composed by a set of `Node` and a set of `Edge`; nodes have a position and a dimension; edges connect a source and a target node. Instances of `Node` or `Edge` are also instances of `Element`, containing an identifier and a type.

An `Element` may also have a temporary status given by an enumerated value, one of `DELETE`, `INSERT` or `MODIFY`. These values are used for stating differences between diagrams,

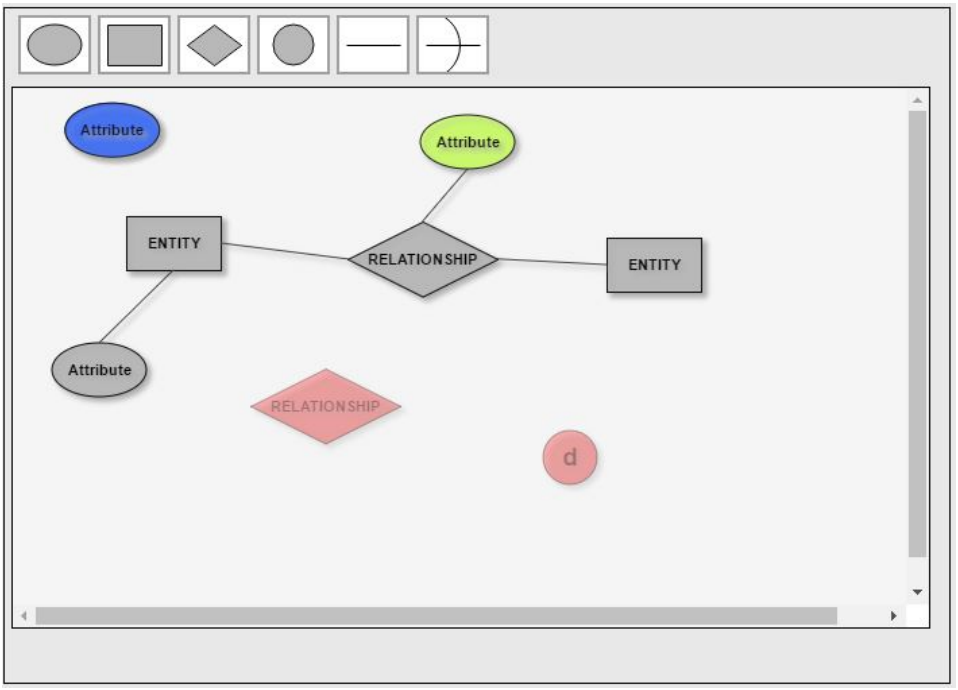

Figure 2 Example of feedback provided by Eshu.

and are used for returning feedback. For instance, an `Element`, either a `Node` or an `Edge` marked as `DELETE` should be removed of the current diagram. Eshu displays in red, blue and light green respectively the elements marked as `INSERT`, `DELETE` and `MODIFY`.

An `Element` may also have a number of instances of `Feature`, which are name-value pairs. Features depend on the type of element, thus cannot be rigidly specified in the schema. For instance, an Attribute is a type of node in an Entity-Relationship diagram which can have different features, such as being a key or multivalued. Thus, an Attribute may have a core graphical representation, a labelled ellipsis, with variants controlled by its features: in a key attribute the label is underlined while a multivalued attribute has a double line ellipsis.

Some features are actually a kind of sub-types. Encoding node characteristics as types or as features is mostly a matter of convenience. One could consider the types "Attribute-Key" and "Attribute-Non-Key" for instance, each with its own graphical representations. Of course, this would lead to a proliferation of types by combining different features that could be rather cumbersome.

Having a large number of types may be relevant for some operations, such as assessing diagrams. Previous research [13] has shown that the assessment of diagrams (represented as graphs) is more efficient when a larger number of types is considered. However, it does not always make sense to use all the features as sub types and it also depends largely on the diagrammatic language being considered. The `reducibles` property of `Diagram` class identifies the names of the types that can be reduced in this fashion, wich ultimately depends on the language.

The diagram serialization is needed by some of the methods provided by the application interface (API) of Eshu. The basic methods are `setGraph()` and `getGraph()`. The former receives a diagram represention as argument while the latter returns one. The method `importGraph()` also receives a diagram serialization but only with differences to the current

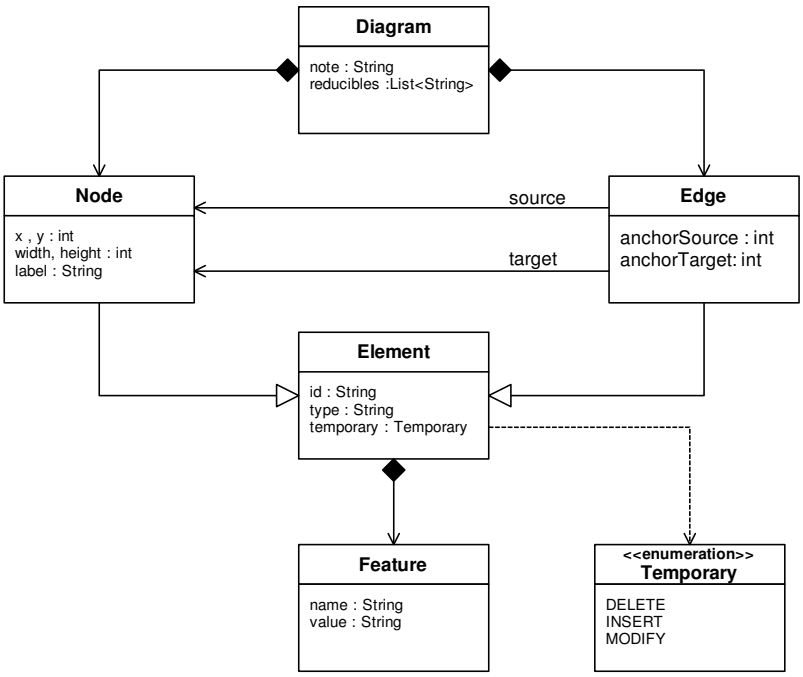

Figure 3 UML class diagram of the diagram serialization used for import/export.

diagram. The diagram serializations imported with this method typically have elements with a `Temporary` value assigned to them and they are displayed accordingly.

The import and export methods are not the only ones in Eshu's API. A partial list of the available methods is presented in Table 1. These methods can be used on an instance of Eshu to configure it. An example of a configuration is the definition of the diagrammatic language, using the `setLanguage()`. An example of binding functions to the user interface would be assigning the copy, cut and paste operations to menu entries or to accelerator keys. As another example, the widgets shown in Figure 1 are all bound to methods of the API.

3.3 Extensibility

Eshu was designed for being extensible, to be able to incorporate new diagrammatic languages, with their own nodes and edges, and enforce restrictions on how these can be connected. These extensions require new JavaScript definitions that have to be integrated with Eshu's source code.

To ease the integration of these definitions, Eshu's code follows an object-oriented approach. JavaScript is not in itself an object-oriented language, but supports to a certain extent a few key concepts of this methodology. For instance, object instances are created with the keyword `new` and a function that acts as a constructor. This function is itself an object with a prototype. The properties of the prototype are shared by all objects it creates, thus these properties can be seen as fields or methods, when they hold values or functions, respectively. It is even possible to simulate inheritance by calling the constructor function of a parent "class".

The new nodes and edges extend abstract definitions provided by the core of the library. These define a number of essential fields, such as position, dimension and label; and also methods for label editing, node resizing, feature selection, moving and dragging, among

Table 1 The application interface (API) of Eshu.

Name method	*Description*
setLanguage	defines the language of the graph
getGraph	returns the graph in json
setGraph	create a new graph using a json text
importDiff	import the differences to the graph
resizeGraph	resize the graph
setPositionHorizontal	change the position of the toolbar to horizontal
setPositionVertical	change the position of the toolbar to vertical
copy	mark to copy a node or a node group
cut	cut a selected node
paste	paste a node or node group that were marked to copy

Listing 1 Entity constructor.

```
function Entity(x, y,id) {
        Vertice.call(this, x, y,id); // extend Vertice
        this.type="entity";
        this.label="ENTITY";
        this.weak=false;
        this.cursorPosition=this.label.length-1;
        this.listTypeCanBeconnected=[
                "attribute", "simpleNode",
                "relationship", "espGenCat"];
}
```

others. However, a few methods have to be explicitly provided, such as the method that displays the element. JavaScript does not have means to enforce the implementation of methods. Thus, mandatory methods are defined by the abstract classes, but the default implementation simply raises an exception unless they are overriden/implemented.

The code snippet in Listing 1 is the constructor of an `Entity` that inherits from `Vertice` all the generic definitions of nodes. The constructor receives a position and an identifier that are passed to the constructor of `Entity`. The constructor also initializes, in this object, specific properties of `Entity`, such as `weak` (by default it is not a weak entity). The last instruction defines a list of node types that can be connected to an `Entity`. Note that in this language two entities cannot be connected directly (only through a `Relationship`), hence the type being defined is absent from this list.

The definition of `Entity` is incomplete without a `draw()` method, as presented in Listing 2. This method receives a graphic context as argument, which the method uses to draw a rectangle, and, optionally, another argument that defines if it is a weak entity or not. Drawing the label at the center is common in most node types, hence this definition can resort to the method `showLabel` inherited from `Vertice`.

The creation of an edge type follows a similar pattern. As shown on Listing 3, the new constructor must extend the `Edge` class which already defines the basic fields and methods required by an edge. Edge takes as argument a source node, a target node, the source anchor index, the index of the anchor and the target node id. It is necessary to define a method `draw()`.

The nodes in the graph are connected with other nodes through edges, but not all nodes can connect with each other. One can set a list of restrictions with objects that check if the

Listing 2 Drawing method of an `Entity`.

```
Entity.prototype.draw = function(ctx) {
        ctx.beginPath();
        ctx.rect(this.x, this.y,this.width, this.height);
        if(this.weak){
                ctx.rect(this.x+2, this.y+2,
                        this.width-4, this.height-4);
        }
        ctx.fill();
        ctx.stroke();

        this.showLabel(ctx);
}
```

Listing 3 Creating a new `Edge`.

```
  function EEREdge1(source,target,handleSource,handleTarget,id) {
        this.links= [
                new SimpleNodeOther(),
                new EntityAttribute(),
                new EntityRelationship(),
                new AttributeAttribute(),
                new AttribuiteRelationship(),
                new RelationshipRelationship(),
                new EntityEspGenCat()];
        Edge.call(this,source,target,handleSource,handleTarget,id);
  }
```

pair of connected nodes has acceptable types. Thus Eshu avoids having links in the graph that are not allowed in the language.

After creating nodes and edges it is necessary to relate them to a language. As shown in Listing 4, the method `setLanguage()` assigns to a language named "eer" a list of previously defined nodes and edge types. This setting has as side effect the creation of a new toolbar in Eshu's user interface.

3.4 Implementation

The implementation of the design described in the previous subsections has some points that need to be detailed. These points are related to the efficiency, automatic layout and integration.

In general, the number of nodes and edges in a diagram is fairly small, since large diagrams are difficult to understand. However, small is relative and for some computers a few dozens of nodes and edges may have impact on efficiency. The most demanding operation from an efficiency point of view is node and edge selection. The iteration over a list of nodes, with a computational complexity of $O(n)$, proved to be too inefficient. Instead, Eshu uses quadtrees, that with a complexity of $O(log(n))$ find nodes and edges more efficiently. The main drawback of using quadtrees for indexing nodes and edges is that it requires reindexing them when they are moved, but the efficiency gain when searching for selected elements pays off.

Listing 4 Defining language EER.

```
 var graph = new Graph(div,700,400);
graph.setLanguage(Language( "err",
        ["attribute","entity","relationship","espGenCat"],
        ["line","lineEGC"]);
```

As a diagram editor, Eshu does not need to layout nodes. The users insert nodes where they please. However, 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. 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 [5]. 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. One of the advantages of a force directed algorithm is that it adjusts to changes, either changes of window's dimension or changes in the number of nodes.

Eshu is a pure JavaScript library, hence it can be integrated in most JavaScipt based web applications. However, some frameworks, such as the 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 for all API methods, such as those listed in Table 1, implemented using the JavaScript Native Interface (JSNI) of GWT.

4 Validation

This section presents an acceptability evaluation of Eshu. To carry out this evaluation, Eshu was integrated in a web integrated environment for learning computer science languages – Enki [9] – to create a diagram assessment module. The assessment of diagrams was provided by a graph-based evaluator [13]. Then, an experiment was conducted with undergraduate students in the laboratory classes of an undergraduate Databases course at the *Department of Computer Science of the Faculty of Sciences of the University of Porto (FCUP)*, from the 1st to the 18th of March, 2016.

Enki is one of the GUIs of Mooshak 2.0 (the new version of Mooshak [6]), a framework for automated assessment of computer science languages. Enki was designed for a wide range of use cases, from introductory high school or college courses, to massive online open courses. It assumes that the students may have little or no help from a teaching assistant and that they may not have the necessary tools installed on their computers. It was developed using Google Web Toolkit (GWT), an open source software development framework that allows a fast development of AJAX applications in Java.

The diagram assessment component compares 2 diagrams using a graph based serialization obtained from Eshu's API. Given two graphs, a solution and an attempt of a student, it computes a mapping between the node sets of both graphs that maximizes the student's grade, as well as a description of the differences between them. These differences are converted to Eshu's diagram serialization and are shown to the student as feedback.

The Enki course created for the experiment contains resources of two types: expository and evaluative. The expository resources are PDFs describing EER languages. The evaluative

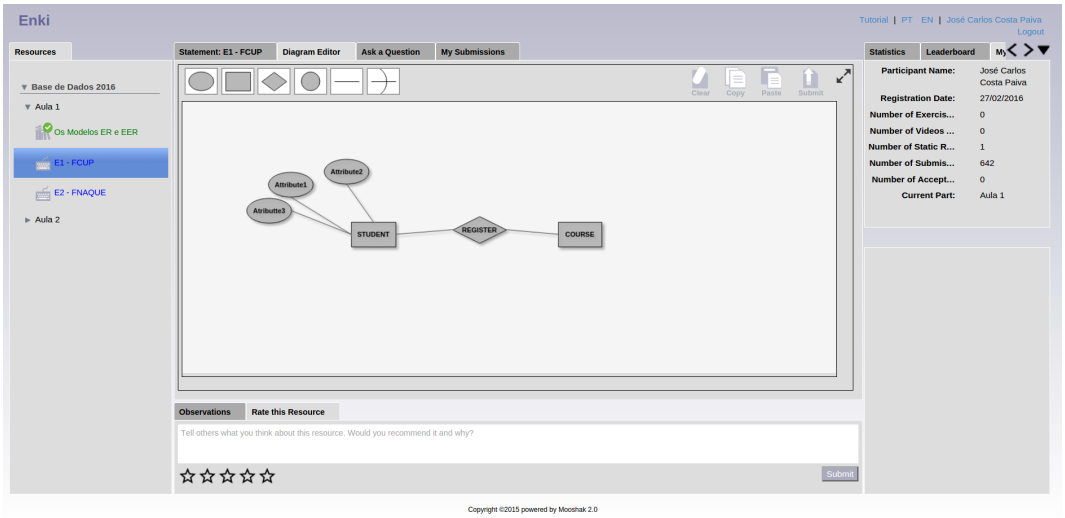

Figure 4 Screenshot of Eshu integrated in Enki.

resources contain the statements describing the database requirements, and an instance of Eshu to create an EER diagram for that database. Figure 4 presents a screenshot of Enki's GUI with Eshu as the editor.

The integration of Eshu in Enki requires a frequent communication between the two, controlled by Enki. Eshu has been designed as a widget, with a binding for GWT, exposing several methods for managing it. An important part of this integration happens when a student sends his diagram for evaluation. Figure 5 presents an activity diagram of this situation.

Firstly, Enki gets the graph from Eshu as a JSON serialization and stores it in a temporary variable. Then, Enki executes an asynchronous request to the graph evaluator to evaluate the graph serialized in JSON. The graph evaluator returns the feedback, observations and a grade of this attempt to solve the exercise. The feedback is a JSON serialization with differences between the attempt and a possible solution to the exercise. These differences are imported temporarily to Eshu, until the student confirms that he is ready to continue from his last attempt. Finally, Enki imports the JSON serialization of the last submission (stored in the temporary variable) to Eshu.

After the experiment the students were invited to fill-in an online questionnaire based on the Nielsen's model [8], using Google Forms. It includes questions on the usefulness of the integration of Enki with Eshu, i.e. on its utility and usability. Utility is the capacity of the system to achieve a desired goal. Usability is defined by Nielsen as a qualitative attribute that estimates how easy is to use an user interface. The survey was completed by 8 students, of which 3 were females.

Figure 6 shows the results grouped by Nielsen's heuristics. The collected data is shown in a bar chart, with heuristics sorted in descending order of user satisfaction.

On the positive side the results prove that the compatibility, consistence and visibility were the heuristics with higher satisfaction. The respondents also selected the ease of use as one of the strongest points of Eshu. On the negative side the results highlighted deficiencies in three areas: speed, reliability and flexibility. Students complained about the difficulty of connecting edges to nodes. This is due to the size of the connecting area of the node. Other students stated that the delay when they validate or submit their programs was too high.

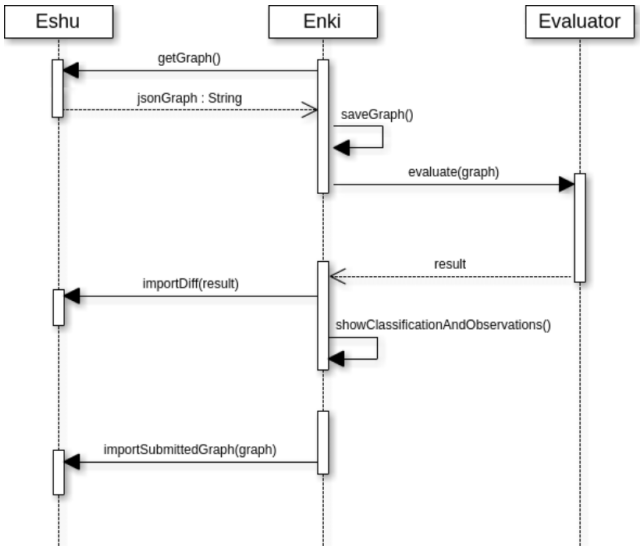

Figure 5 Communication diagram of Eshu with Enki upon a submission to the evaluator.

Most of these issues were already fixed as result of the students' feedback.

The questionnaire finalizes with an overall classification of Eshu in a 5 values Likert-type scale (very good, good, adequate, bad, very bad). Many of the students (37.5%) classified Eshu as an adequate tool and others (37.5%) stated Eshu as a good or a very good tool. Some students (25%) found it either bad or very bad.

5 Conclusions

The creation of a language based environment requires an editor. Many editors are available for textual languages, but the offer is much more limited for diagram editors, in particular for embeddable in web based e-learning environments. Eshu is a JavaScript library that offers diagram editing to web based applications, with an emphasis on the production of valid diagrams within a given language, and in the interoperability with other software components.

Eshu was designed for interoperability and extensibility. It has a diagram definition language for data interchange with other components, which includes complete diagrams and diagram differences used for reporting feedback. This data is serialized in JSON and its schema is formalized in JSON Schema. Although implemented in JavaScript, Eshu follows an object-oriented methodology, which simplifies the introduction of new node and edge types by extending core classes.

To validate the proposed library, Eshu was integrated in Enki, the pedagogical interface of Mooshak 2.0. Eshu is one component of the diagram language module, where it interacts with a graph based evaluation component. This integration validated the interoperability features of Eshu and was also the base for an usability evaluation. To complete the validation of the extensibility features, new languages must also be supported.

The candidates to new languages in Eshu are deterministic finite automata (DFA) and UML diagrams. The former should be straightforward for Eshu (although more demanding from an evaluation point of view). UML diagrams are a bit more complex due to their variety and profusion of features, particularly in class diagrams.

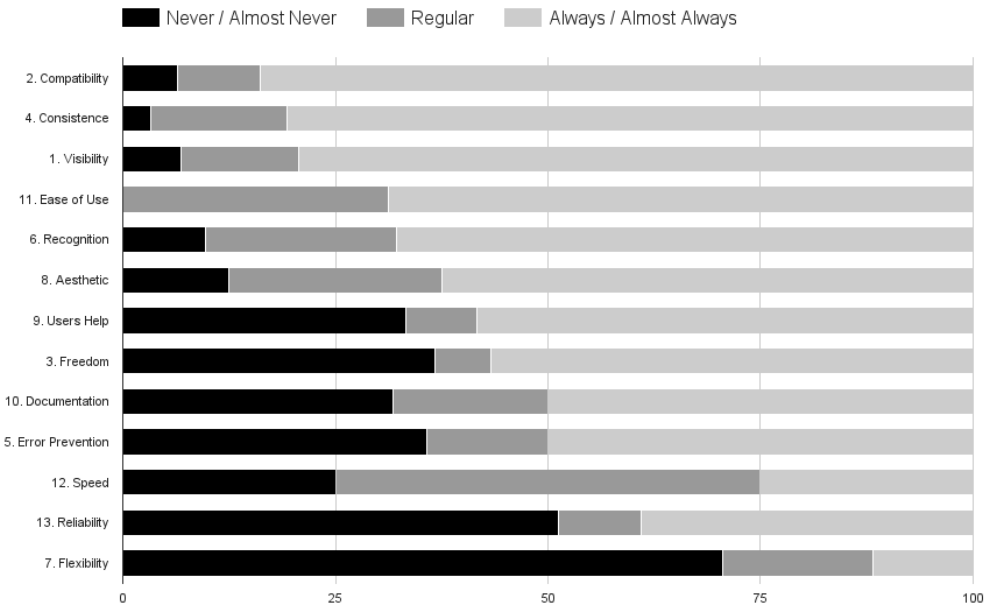

Figure 6 Eshu acceptability evaluation.

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 *Proceedings of the Twenty-Third international joint conference on Artificial Intelligence*, pages 1976–1982. AAAI Press, 2013.

3 Firat Batmaz and Chris J. Hinde. A diagram drawing tool for semi–automatic assessment of conceptual database diagrams. In Myles Danson, editor, *10th CAA International Computer Assisted Assessment Conference*, pages 71–84. Loughborough University, 2006.

4 Kris Zyp Francis Galiegue and Gary Court. Json schema: core definitions and terminology. Technical report, Internet Engineering Task Force, 2013.

5 Stephen G. Kobourov. Spring embedders and force directed graph drawing algorithms. *CoRR*, abs/1201.3011, 2012. URL: `http://arxiv.org/abs/1201.3011`.

6 José Paulo Leal and Fernando Silva. Mooshak: a web-based multi-site programming contest system. *Software: Practice and Experience*, 33(6):567–581, 2003. `doi:10.1002/spe.522`.

7 JGraph Ltd. Build interactive web diagramming apps. Accessed: 2016-03-18. URL: `https://www.jgraph.com/`.

8 Jakob Nielsen and Thomas K Landauer. A mathematical model of the finding of usability problems. In *Proceedings of the INTERACT'93 and CHI'93 conference on Human factors in computing systems*, pages 206–213. ACM, 1993.

9 José Carlos Paiva, José Paulo Leal, and Ricardo Queirós. Enki: A pedagogical services aggregator for learning programming languages. (in press), 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 Northwods Software. Gojs – interactive diagrams for javascript and html. Accessed: 2016-03-18. URL: `http://gojs.net/`.

12 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.

13 Rúben Sousa and José Paulo Leal. A structural approach to assess graph-based exercises. In José-Luis Sierra-Rodríguez, José-Paulo Leal, and Alberto Simões, editors, *Languages, Applications and Technologies*, pages 182–193. Springer International Publishing, 2015. doi:10.1007/978-3-319-27653-3_18.

14 Vinay Vachharajani and Jyoti Pareek. A proposed architecture for automated assessment of use case diagrams. *International Journal of Computer Applications*, 108(4):35–40, December 2014. Full text available.