A Comparison of Client-Side Web Diagramming Libraries for Generating Modeling Editors

Andreas Huber

Technische Universität Dresden, Dresden, Germany
andreas.huber2@mailbox.tu-dresden.de

Abstract. Modern client-side diagramming libraries that can be executed in the browser offer a broad range of functionality to create, display and manipulate graphs and diagrams. This work compares 14 libraries in five categories. Business aspects like license and active maintenance, size and dependency aspects, customizability aspects, as well as user experience and developer experience aspects are discussed. The results are presented in a tabular overview to facilitate a convenient high-level comparison of the libraries. Every library is then discussed briefly at a high level with a focus on the customizability of graphical elements. The overview provided by this work can assist in assessing suitable candidates for diagramming projects. Following this comparison, three preferred libraries for implementing editors for models conforming to metamodels specified in Ecore/EMF are identified.

1 Introduction

There is proverbial wisdom in the saying *"a picture says more than a thousand words"*. When conveying information, a graphical representation can be more intuitive to understand than a textual representation. An example for this approach is the realm of graphical specification and modeling languages like the *Business Process Model and Notation* [OMG11] or the *Unified Modeling Language* [OMG17], where graphical representations of e.g. processes or concepts specify, document, and communicate knowledge.

For graphical modeling languages like these or others e.g. specified with the help of metamodels, editors for viewing and manipulating models are a necessity. With the rise and broad availability of web technologies providing possibilities for a more graphically interactive experience like HTML5 [Gro21] with e.g. its canvas or seamless Scalable Vector Graphics (SVG) integration, web applications provide an attractive development target for offering client-side graphical modeling tools for a broad range of users without having to implement platform-specific solutions. On top of that, an extensive variety of tooling provides abstraction layers to facilitate the development of graphical editors for modeling purposes. This work aims at comparing suitable candidates for creating such client-side editors and giving an overview about them for developers, which plan to design and implement a modeling application.

Section 2 introduces inclusion criteria for existing solutions. The resulting list of libraries is then presented. Section 3 specified an analysis scheme for the discussed tooling. The main part of this work is Section 4, which presents the results of the comparison and briefly describes the individual libraries. Based on this comparison, three candidate libraries are identified in Section 5to be used in a project which aims at automatically generating client-side graphical editors for models conforming to specified metamodels in Ecore. This overview paper is then concluded in Section 6 with a summary.

2 Inclusion Criteria for the Comparison of Graph and Modeling Libraries

This section defines criteria, which possible libraries have to meet to be included in the analysis. To render an environment-agnostic view of the landscape of diagramming libraries, only client-side libraries are considered. This work understands client-side libraries as those, that don't need to be integrated into other (e.g. front-end or back-end) frameworks and are able to run standalone. Those libraries offer the benefits of being executable in virtually any browser environment with or without being integrated into larger applications.

Furthermore, graphical models usually comprise visually connected elements to describe a certain structure and can be represented with graph-like structures. Therefore, only libraries will be considered, which offer this functionality - drawing connections between rendered elements, resembling graphs.

This results is the following list of libraries: *Diagram.js, GoJS, JointJS, Rappid, mxGraph, JSPlumb Community & Toolkit Edition, Draw2d JS, Cytoscape.js, D3, DHTMLX Diagram, Mindfusion Diagram Library,* and *yFiles for HTML.*

3 A Classification Scheme for Comparing Graphical Editor Libraries

For giving a methodological comparison, a classification and analysis scheme is applied. Five categories analyzing 17 different aspects pose the framework for comparing the found libraries and finding suitable candidates for different use cases. Different viewpoints on a software project have to be taken into consideration when choosing libraries for implementing features. Business aspects, as well as concerns from developers and users, need to be regarded. Therefore, the classification is split into the following categories with each one having several concrete facets to identify and evaluate:

- **Business aspects**: besides technical aspects, peripheral characteristics like licensing and the library's maturity are a factor which can weigh in heavily in decisions for or against a library.
  - *License*: how is the library licensed?

  - *Version number*: version numbers may indicate the library's maturity, but since version numbers can be chosen at the library owner's own discretion, this information has to be evaluated carefully.
  - *Last release*: when was the last release?
  - *Under active development*: is the library under active development? Active development is classified as contributions to the source code of the project or releases in the 12 months.
  - *Continuous integration (CI) pipeline*: is there a continuous integration pipeline for the project, which indicates a systematic approach in finding bugs, regression, and compilation or runtime errors?
- **Size**: since client-side libraries have to be served to the user, the size of a library can be an important factor for the decision for or against it. Furthermore, a high amount of dependencies may bear a higher risk of deprecation, incompatibilities, and security issues.
  - *Package size*: how large is the released library in Kibibytes including all dependencies? For every library, the size of the minified distribution is analyzed. If no minified distribution is provided, the size of the regular distribution is analyzed. This is marked with an asterisk (*) in the comparison in Table 1, which can be found in the appendix.
  - *Number of direct dependencies*: how many packages depends the library directly on? This information is gathered from package manager files or if missing, from the website of the project.
- **Customizability**: graphical editors can be implemented for different use-cases, e.g. for modeling based on metamodels. Metamodels can differ heavily, as can their intended graphical representation. Editors need to be customizable to a certain degree to fulfill different requirements.
  - *Custom nodes*: can custom graphical representation of nodes or elements as well as compounds be specified?
  - *Custom edges*: can connections between elements be customized?
  - *Easy customization*: how effortlessly can the graphical customization and complex figures be implemented? This category assesses whether the shape and styling of graphical elements in a diagram can be defined or customized easily. An explanation for the decision is given in the more detailed portrayals of the libraries.
- **User experience**
  - *Native palette & toolbar support*: are toolbars or palettes with blueprints of graphical elements natively supported?
  - *Drag & drop*: can the position of graphical elements like nodes and connections be changed via drag and drop?
  - *Undo & redo*: is there support for undoing and redoing user actions?
- **Developer experience**: this category shines a light on how demanding working with a certain library is for a developer.
  - *Comprehensive documentation*: is there a comprehensive and up to date documentation for the library's core principles and API?
  - *Examples*: are there code examples making it less cumbersome for a developer to get familiar with the library?

- *Serialization*: is serialization and de-serialization of the resulting graphs and diagrams supported natively by the library?
- *Implementation language*: in which languages is the library implemented?

4 Results and a Brief Analysis of the Compared Libraries

The analysis scheme defined in Section 2 is applied to the found libraries. An overview of the results displayed in a tabular overview facilitating a convenient high-level comparison of the libraries can be found in Table 1 in the appendix. The following subsections describe each library from a high-level view, trying to shine a light on the basic concepts and especially customization possibilities for creating complex graphical elements.

4.1 GoJS

GoJS[1] is a commercial library that can be used to develop interactive graph editors as well as visualize data or drawing tools. Licensing costs start at 3.495$. It offers a clear separation of model and view, where nodes and edges are defined independently of their graphical representation, which can automatically layout the resulting graph without overlapping elements, if possible.

Custom shapes can be defined easily by either composing and stacking the provided basic geometric shapes or by importing SVGs. The resulting node and edge templates can be data-bound to user-defined model properties with a slot-like syntax, making it convenient to specify new types of elements.

Events are emitted and communicated by a command handler, which manages undo and redo operations, too.

4.2 JointJS

JointJS[2] is a library developed for creating, displaying and editing graphs. Nodes are created as either basic or custom shapes. Their position has to be specified as well. It is defined relative to its containing element in the DOM. Optionally, Ports for the ends of connections between nodes can be defined.

Dragging and dropping from one node's port to another can connect nodes. A magnetic behavior, where the connection ends snap to other ports is optional.

Custom nodes can be composed of SVG and its common geometric shapes. Edges connecting nodes can be customized as well. CSS-styling can be applied. Edges have source markers and end markers, which are displayed graphically and can be customized as well.

An event system is implemented, where interaction with the graphical representation as well as changes in the data model are communicated.

[1] https://gojs.net/latest/index.html, 13.01.2021
[2] https://github.com/clientIO/joint, 13.01.2021

4.3 Rappid

Rappid[3] is a commercial extension for JointJS. Licensing costs start at 1500€. It extends JointJS with support for various graph drawing and editing and infrastructure services:

- a control panel for editing and a palette for element creation
- undo and redo support
- export to graphical bitmap formats
- automatic layouting of graphs
- inline text editing of node and connection labels
- a properties editor for manipulating nodes and edges

4.4 D3

D3[4][5] is a library whose main purpose lies in data visualization, not directly in drawing and editing graphs. Its basic concept lies in the binding from data to DOM elements, most prominently SVG or HTML elements. Elements are selected and then manipulated according to the provided data and binding from parts of the data to concrete elements or their attributes. This mechanism can be exploited for creating nodes and edges for graph data, which is to be visualized.

Furthermore, D3 offers support for network visualization with e.g. force-directed layouting of graphs. The library's support for this focuses on visualization, too. Customization possibilities are therefore limited.

4.5 Dagre-D3

Dagre-D3[6] provides an abstraction layer on top of D3 and employs a graph representation library for directed graphs.

The library is question is Dagre[7], which is a tool for computing a layout for directed graphs. Input for an autolayouting task consists of a specification of the nodes and connections between nodes and dimensions in terms of height and width of each node as well. The resulting layout is represented by a spatial position for each node and points, where connections intersect with nodes.

The actual rendering of a layout is then implemented with other libraries or approaches. This is where Dagre-D3 comes into play. Graphs are rendered with D3. Dagre-D3 provides an abstraction layer for generating shapes and their visual representation. Compounds can easily be specified as well.

[3] https://www.jointjs.com/, 15.01.2021
[4] https://github.com/d3, 19.01.2021
[5] https://d3js.org/, 19.01.2021
[6] https://github.com/dagrejs/dagre-d3, 04.01.2021
[7] https://github.com/dagrejs/dagre, 30.01.2021

4.6 Diagram.js

Diagram.js[8] is a library for drawing and editing diagrams with a focus on extensibility. It is designed as a base library for BPMN.iO[9] but can be adapted to other use cases due to its modular nature.

It provides a data model for graphs with support for parent-child-relationships between nodes. Extendable element- and graphics factories are responsible for creating nodes and their graphical representations. A node management system manages the lifecycle of graph elements. Changes are propagated and un- and re-doable via an event bus. Natively supported palettes can be customized as well. Rules can be defined for allowed and forbidden interactions and connections between types of nodes and connections.

Customizability is further supported by the module system of Diagram.js. Diagram components are discovered and connected via dependency injection. This can be leveraged for customizations for an editor and as a base for product lines of editors.

4.7 mxGraph

mxGraph[10] is a diagramming library most prominently used in diagrams.net[11]. It is one of the oldest of the analyzed libraries and has officially reached an end-of-life state with no support or further development as of November 2020. It is a general-purpose diagramming library.

Nodes are specified including their spatial position, width, and height as well as a possible hierarchical relation to other nodes. If shapes exceeding basic geometric primitives are to be constructed, they have to be specified relatively cumbersome with paths in a relative coordinate system as well as with SVG markup abstracted through an API.

mxGraph offers a transaction system for maintaining a consistent model state and, compared to other libraries, a rather low-level API with a broad range of management and grouping functionality.

Unfortunately, the documentation contains partly deprecated data.

4.8 JsPlumb Community Edition

JsPlumb Community Edition[12][13] is a toolkit for visualizing connections between elements on web pages. Existing DOM-element are made manageable by JsPlumb to facilitate dragging, dropping, and connecting. Connections are automatically created via SVG. They are connected to anchors on the managed DOM elements which serve as nodes. Anchors can be specified relative to the element,

[8] https://github.com/bpmn-io/diagram-js, 13.01.2021
[9] https://bpmn.io/, 13.01.2021
[10] https://github.com/jgraph/mxgraph, 15.01.2021
[11] https://www.diagrams.net, 15.01.2021
[12] https://github.com/jsplumb/jsplumb, 17.01.2021
[13] https://docs.jsplumbtoolkit.com/community/current/index.html, 17.01.2021

for which they serve as a port. Connecting nodes is realized by dragging a line from a port to another port. Rules can be specified, which types of anchors are allowed to be connected with other types.

This approach offers the advantage, that nodes can be specified and styled purely with HTML, SVG, or other specification languages that create elements.

4.9 JsPlumb Toolkit Edition

JsPlumb Toolkit[14] is a commercial extension for JsPlumb Community Edition. Licensing costs start at 1995£. It extends it by adding a separation of data and view. Nodes and edges can be specified in a model, which then is rendered as a graph. Node templates can be specified in HTML with variable slots for data binding. A palette then offers functionality to drag and drop new elements on the canvas. Nodes and edges can be edited with dialog support. Serializing and deserializing data is supported natively in comparison to JsPlumb Community Edition.

4.10 Draw2d JS

Draw2d JS[15] is a tool to create drawings and graphics resembling those created with Microsoft Visio. Besides charting, it provides the functionality to compose diagrams from basic geometric figures. A high-level API serves as a proxy for drawing and composing these basic shapes. In comparison to most other libraries, layouts can be explicitly specified, e.g. rectangles with labels in layouts like flow layout or vertical layout, which dictate the appearance of the composition.

Nodes can then be connected with graphically customizable connections. The intersection between a node to which the relation is connected and the connection itself can be decorated with symbols. Ports on nodes can be specified as well.

Advanced features like snapping to grids or other shapes on dragging and dropping, selection feedback or a command stack with whose help undoing and redoing of commands can be implemented are available, too.

4.11 Cytoscape.js

Cytoscape.js[16] is a graph library for graph analysis and visualization. It aims at supporting graph analysis with graph theory methods. Nodes are specified with an initial position and can have custom properties. The traditional graph model is extended by compound nodes, where nodes can have a parent-child-relationship indicating a hierarchy.

Computation on the graph model can be done, e.g. searching for shortest paths or clustering. The focus lies on the analysis of and computation on the specified, not on the editing of created graphs.

There is just limited support for customizing the visual appearance of nodes and edges as well as limited support for direct editing.

[14] https://jsplumbtoolkit.com/features, 17.01.2021
[15] https://github.com/freegroup/draw2d, 16.01.2021
[16] https://js.cytoscape.org/, 19.01.2021

4.12 DHTMLX Diagram

DHTMLX Diagram[17] is a commercial library for creating diagrams with predefined shapes and connectors and own diagrams. Licensing costs start at 599$.

A model represents nodes and connections. Nodes are specified with positions, a type, and a label as well as user-defined properties. Edges connect nodes, have a type and two customizable ends. Custom shapes can be specified with the help of SVG or HTML templates with variable slots, where data from the user-defined properties is bound to.

4.13 Mindfusion Diagram Library

Mindfusion Diagram Library[18][19] is a port of the native diagramming framework Mindfusion[20]. Mindfusion is a collection of UI elements like schedulers, reports, maps, spreadsheets, charts or diagrams. Licensing costs for the JavaScript diagramming library start at 599$.

The graph model just contains information about the nodes and edges omitting data for graphical representation and therefore enforcing a strict separation of model data for conceptual data and view data. Nodes are specified with an ID and custom properties, links connect nodes via their IDs.

Nodes themselves can display images and multimedia data like images and videos. Custom nodes can be composed of classes like basic shapes, SVG, composite, table, free-form, or container.

4.14 yFiles for HTML

yFiles for HTML[21] is an elaborate commercial library for visualizing diagrams. Licensing costs start at 18.200$. It provides functionality for the creation and interactive viewing and manipulation of graphs.

Multiple data sources can specify graphs, e.g. graphs specified in GraphML. Templates for nodes with variables for node properties can be specified in HTML or SVG. This library provides a broad range of node as well as edge visualizations out of the box. Both ends of edges can be customized, too.

Interactive aspects are e.g. displaying a varying level of rendered graph details according to the zoom level, or grouping or filtering on graph elements.

Layouting is calculated automatically as well as efficient edge routing.

[17] https://docs.dhtmlx.com/diagram/overview.html, 20.01.2021
[18] https://www.mindfusion.eu/onlinehelp/jsdiagram/index.htm, 13.01.2021
[19] https://https://mindfusion.eu/javascript-diagram.html, 13.01.2021
[20] https://mindfusion.eu/index.html, 13.01.2021
[21] https://www.yworks.com/products/yfiles-for-html, 13.01.2021

5 A Selection of Libraries for Generating Editors for Models Based on Metamodels

This work is motivated by a project which tries to automatically generate lightweight and client-side graphical web diagramming editors for models conforming to given metamodels specified in Ecore, an integral part of the Eclipse Modeling Framework (EMF)[22]. The project aims at enabling users of the generated editor to generate valid models. These models then can be serialized and seamlessly imported and used again in the Eclipse tooling as if they were created natively in Eclipse with e.g. the generated table editors in an *Edit*-Plugin.

The need for a suitable diagramming library prompted this systematic comparison. Based on the collected data, candidate libraries for the implementation are to be identified and then further evaluated in follow-up work. Basic requirements for the diagramming library are the following:

1. The chosen library has to be **open source**.
2. There is no uniform way of displaying concepts and relations described by a metamodel. Therefore, shapes for representing them need to be **as customizable as possible with as effort as possible**.
3. Since user-created diagrams have to be transformed to instances of a metamodel in the EMF-universe, the library should support a **straightforward serialization** of the diagrams.

Candidate library fulfilling these requirements are *JointJS*, *Diagram.js*, and *JsPlumb Community Edition*. All three are licensed with permissive licenses.

Concerning customizability, JointJS, as well as Diagram.js, define custom shapes with the composition of SVG shapes. In contrast, JsPlumb does not natively support the construction of nodes but focuses on visualizing connectivity of rendered elements of a DOM. This enables a more intuitive way of describing structured data and their graphical representation with HTML as well as SVG elements, which can be styled with CSS, too. This may pose a lower entry barrier and higher expressiveness for visualizations.

The ease of serializing the created diagrams differs in all libraries. JointJS serializes its diagrams natively to JSON objects, which describe the position, type, and properties of each node as well as their connections and their types. Diagram.js offers a minimalistic way to export the structure of the models to JSON. As has been mentioned, bpmn.io employs Diagram.js for diagramming. BPMN models are serialized with a dedicated plugin transforming the internal model to an XML while validating the model against the BPMN metamodel. This additional layer introduces further complexity but shows potential for the use case of the described project, where models need to conform to a specified metamodel. JsPlumb does not support the serialization of diagrams out of the box. This feature has to be implemented by iterating over and analyzing elements in the diagram container. This is a major drawback compared to the other libraries, which support this without additional implementation effort.

[22] https://www.eclipse.org/modeling/emf/, 30.01.2021

In conclusion, Diagram.js is the most promising of the compared libraries for the described use case, especially due to its extensible nature and plug-in architecture. This library provides the most benefit in the category of user experience aspects, too. Nevertheless, one major drawback is the lack of high-level documentation, which complicates the conceptualization and implementation. This drawback impedes the decision to designate a preferred library for this project, leaving it to a more thorough analysis to make a final decision.

6 Conclusion

As can be seen in Table 1, there's a broad range of libraries covering a broad range of functionality from simply displaying graphs over offering a full-fledged diagramming software development kit to diagramming being just a part of the functionality of a broader framework of components. The overview provided by this work can assist in assessing suitable candidates for diagramming projects.

Three candidates are identified as possible libraries for serving as the diagramming foundation in a project for generating editors for models based on metamodels. The libraries in question are *JointJS*, *Diagram.js*, and *JsPlumb Community Edition*.

References

Gro21. WHATWG Web Hypertext Application Technology Working Group. HTML Living Standard. https://html.spec.whatwg.org/, January 2021.

OMG11. Object Management Group OMG. Business Process Model and Notation (BPMN). http://www.omg.org/spec/BPMN/2.0, January 2011.

OMG17. Object Management Group OMG. Unified Modeling Language® (OMG UML®). https://www.omg.org/spec/UML/2.5.1/PDF, December 2017.

A Appendix - Tabular Overview Over the Compared Libraries

Table 1. Comparison of the found libraries. An asterisk (*) marks not minified package sizes

		JointJS	Diagram.js	JsPlumb (Community)	JsPlumb (Toolkit)	GoJS	Rappid	D3	Dagre-D3	mxGraph	Draw2dJS	Cytoscape.js	DHTMLX	Mindfusion	yFiles
Business decisions	**License**	MPL 2.0	MIT	MIT/GPL	Commercial	Commercial	Commercial	BSD-3-Clause	MIT	MIT	MIT	MIT	Commercial	Commercial	Commercial
	Version	3.2.0	4.0.0	2.15.5	2.4.2	2.1.3.4	3.2.0	6.3.1	0.5.0	4.2.2	1.0.38	3.17.1	3.0.3	3.5.2	2.3.0.4
	Last release	6/2020	7/2019	12/2020	no info	1/2021	no info	12/2020	12/2017	10/2020	7/2020	12/2020	12/2020	no info	no info
	Under active development	✓	✓	✓	✓	✓	✓	✓	✓	✗	✓	✓	✓	✓	✓
	CI pipeline	✓	✓	✗	no info	✗	no info	✗	✗	✗	✗	✓	no info	no info	no info
Size	**Package size [KB]**	405	576*	211	no info	897	1327	262	708	792	2474*	106*	295*	1006	no info
	Number of direct dependencies	5	9	0	no info	0	8	30	4	0	2	2	0	no info	no info
Customizability	**Custom nodes**	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
	Custom edges	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
	Uncomplicated customization	✓	✓	✓	✓	✓	✓	✗	✓	✗	✓	✗	✓	✓	✓
User experience	**Native palette or toolbar support**	✗	✓	✗	✓	✓	✓	✗	✗	✓	✗	✗	✓	✓	✓
	Drag & drop	✓	✓	✓	✓	✓	✓	✓	✗	✓	✓	✓	✓	✓	✓
	Undo & redo	✗	✓	✗	✓	✓	✓	✗	✗	✓	✓	✗	✓	✓	✓
Developer experience	**Comprehensive documentation**	✓	✗	✓	✓	✓	✓	✓	✗	✗	✗	✓	✓	✓	✓
	Code examples	✓	✓	✓	✓	✓	✓	✓	✗	✓	✓	✓	✓	✓	✓
	Serialization	✓	✓	✗	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
	Implementation language	JS (JavaScript) TS (TypeScript)	JS	JS/TS	JS/TS	JS/TS	JS/TS	JS	JS	JS	JS	JS	JS/TS	JS/TS	JS/TS