# PlayIt: A Pluggable And Interactive Drawing Tool For Technical Applications

Nikitha Rao, Rutha Prasad, Pallavi M.P., Nithin Bodanapu, Reshma Bhat, N.S. Kumar,
PES University
Bangalore, Karnataka, India

## ABSTRACT

Diagrams and visualizations are predominantly used to convey concepts and ideas, especially in a classroom setting. While there are several online tools available that facilitate construction of technical diagrams, there is no single universal tool that allows users to both draw and interact (evaluation of a circuit) with a given diagram that spans over multiple areas of interest (or domains). In our paper, we propose novel techniques that can be incorporated into drawing tools that allow an expert user to plug-in their domain of interest by writing a domain specific script. By doing so, we give flexibility to the expert to add additional features and domains without having to make changes to the tool. The tool can be used for drawing technical diagrams, analyzing them and evaluating them based on rules, in the form of predefined functions, provided by the expert user. We also introduce the concept of using formal representations for diagrams rather than storing them as images, thereby making it easier to analyze and manipulate. Additionally, we allow the user to import diagrams drawn on other platforms and interact with them. As a proof of concept, we developed a web-based drawing tool called PlayIt which supports the domains digital logic circuit design, finite state automaton and basic geometry. The tool's architectural design caters to the limited programming experience of non-computer science major students and teachers. We also present several applications of the tool in aiding the teaching-learning experience such as automated evaluation of assessments, bringing the focus on understanding concepts rather than functioning of multiple tools etc. A user study supported our claims with several positive responses from both students and teachers.

## KEYWORDS

drawing tool, interactivity, web, plugin, education, automated assessment, teaching aid

**ACM Reference Format:**
Nikitha Rao, Rutha Prasad, Pallavi M.P., Nithin Bodanapu, Reshma Bhat, N.S. Kumar, . 2019. PlayIt: A Pluggable And Interactive Drawing Tool For Technical Applications. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn


*Conference'17, July 2017, Washington, DC, USA*

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

## 1 INTRODUCTION

Diagrams have always been a primary choice of visual aid to simplify complex ideas. While there are several tools available that help facilitate construction of such diagrams, there is no single universal tool that spans over multiple domains (areas of interests) that may be relevant to a user, especially for students. In addition to this, when it comes to technical diagrams, there may be several ways in which a user may want to interact with the diagram, for example - evaluating a logic circuit given some inputs. It is not uncommon for users to want to analyze the diagrams and evaluate them. While there are some tools which allow these interactions, they are restricted to a specific domain and offer a selected set of interactive functions.

Driven by the need for such a extensible tool that is flexible enough to meet the varying needs of different users and domains, we designed a web-based drawing tool for technical applications. As the title suggests, PlayIt is a tool that can be plugged in with any domain and configured to meet the needs of a given user without making any changes to the tool itself, thus supporting extensibility. It also brings the diagrams to life and allow users to interact and "play" with it ( design details discussed in Section 4). This gives all expert users (teachers or users who have extensive knowledge in the domain area) the choice to add any domain to the tool and make it available to other users. This is achieved by allowing the expert user to add in their domain-specific scripts, which can be started off from our basic script template, directly into the tool (discussed in depth in Section 5).

As mentioned earlier, just drawing a diagram may not always be enough. Taking this into consideration, we let the domain expert users decide what interactions they think is useful and relevant to the domain of interest. The domain-expert users are given flexibility to decide on the interactivity required for their target users. The interactions can be as simple or complex as they desire. This is especially useful when teachers want to control what interactions and functions should be available to students when they are required to create the figure and interact with it relevant to some concept being taught.

One way to evaluate the correctness of diagrams in a test setting could be to use image processing techniques. This is not only extremely complex, but also very dependent (on a domain) and unreliable. To help solve this problem, we introduce the concept of using formal representations for diagrams. This also makes evaluation easier as each interaction is now the result of predefined functions that manipulate the diagram's representation rather than the diagram itself. We have defined the formal representation used to store the figures in Section 3. This representation can be transformed if required by the expert user. The expert of the domain also defines the functions thus allowing the interactions to be more robust and

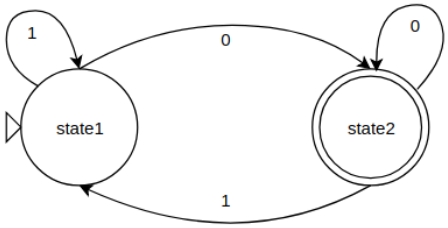


**Figure 1: A drawable image, a deterministic finite state automaton that accepts even binary numbers**

| - | state1 | state2 |
|---|---|---|
| state1 | 1 | 0 |
| state2 | 1 | 0 |

**Table 1: The corresponding formal representation of the DFA in Figure 1 would be a transition table. The start state being state1 and state2 being the final state.**

accurate. Figure 1 is an example diagram of a deterministic finite state automaton which accepts even binary strings. The formal representation for the same can be transformed to a transition table. Table 1 shows the transition table for Figure 1. Using this formal representation in place of images helps us overcome several challenges that arise in image processing and makes the system invariant to scale and rotation, which are difficult to tackle using traditional image processing. In addition to this, image processing techniques are more prone to errors and are not very robust. Though using machine learning or deep learning can help overcome some of those problems, it may be harder to get the desired accuracy in analyzing and evaluating diagrams. This is important as we target automatic evaluation of answers as an application for the tool.

We have also provided a mechanism to allow users to load their diagrams that may be stored in XML or other non-image formats from other tools. From a student's (naive user's) perspective, the user can load their diagram and interact with the diagram based on the interactions provided by the tool. From a teacher's (expert user's) perspective, there may be certain functions that they'd want to carry out which other tools may not provide. The teacher can simply add that functionality to the tool, demonstrate the same and make it available to the students to use. We have demonstrated the same by using JFLAP [12]. Despite this being a one way connection for now, it can easily be made two way by allowing other tools to call functions on the tool and displaying the results obtained on their platform.

The key technical contributions made in our paper is as follows:

- Extensibility: The tool can be augmented with any domain of interest making it programmable and generic.
- Interactivity: We introduce the concept of using formal representations for diagrams instead of storing them as images. This enables users to interact with the diagram in various ways.
- Pluggable: The domains and operations added are essentially plug-ins as they can add functionality to the tool without making any changes to the tool itself.
- Services: A user can import and interact with diagrams from other tools. The operation functions themselves act as APIs as other tools can effectively make use of the interactions that are defined in the tool.
- PlayIt: Tool developed to demonstrate all the ideas presented in our paper. As a proof of concept, three domains, namely, digital logic circuit design, finite state automaton and basic geometry were added to the tool. Several interactive functions were also added to each one of the three domains.

The organisation of the remainder of the paper is as follows: Section 2 identifies the limitations with various other tools that are currently available. The formal representation that is used to store the diagrams is described in Section 3. The overall design of PlayIt, the tool developed by us and the implementation details are enclosed in Section 4. Section 5 enlists the algorithms for extending the tool with different features. This is to emphasise on the simplicity of the whole process. The current domains and interactive features that were added to the tool as proof of concept have been enlisted in Section 6. This is immediately followed by the evaluation of the tool in Section 7. We conclude with Section 8 and 9 which give a brief overview of the applications and the future work respectively.

## 2 RELATED WORK

To the best of our knowledge, current education platforms do not support extensibility by offering adaptation of customized code modules. Analyzing and supporting alien functionalities requires architectural diagnosis and interfacing, which are heavy design implementations. For education in particular, tools have been used in training specific courses only, restraining the involvement of instructors and students in customizing their own work flows and automated assessments. Studying existing works helped design the analytical framework that uses abstract interpretation to allow instructors to scale and students to learn in multi-domain training.

**Pre-Packaged libraries of multiple domains with no assessment support -** Training tools, in our use case, graphic training tools, are generic to drafting and layouts in multiple domains in order to improve the resource-fullness and speed of creation of standard graphic elements [8][15]. Main aim being to displace manual technical drawing, and have common utilization as the main tool in computer-aided drawing, drafting and design like Draw.IO [7], Vectr [19]. With no ability to interact with or analyze the diagrams. Analytical platforms would require understanding new user input [18] and customizing output behaviour accordingly.

**Trade off between offering zero customizability but high specific functionality -** Certain tools having fixed their functional context to a small domain, can provide it's extensive simulated functionalities. Proving to be very powerful in their respective domains, the architectures are not built to be extendable. Domain-specific tools, Automaton Simulator [3] and JFLAP [12] dedicated to finite automate courses offer complex pre-packaged functionalities which limit the tool to a specific domain. Deciding on the

trade-off between specificity and generic modules helps design synthetic architectures. Studies on the need to ease-in students into new development environments [6] where they can find familiar functionalities as taught during the course indicate the need for adapting functionalities based on user feedback.

**Offering adaptation of only low complexity alien code** - As an objective to allow custom interactions on diagrams, some solutions allow adding functionality externally, but are required to keep the functions rudimentary and specific to fit the back-end code structures that they run on [11]. Adobe Illustrator [1] is known to have a C plugin that can be used to develop custom simple functionalities. To allow instructors to add in their own customized assessments would only require mastering of formal verification techniques and implementing generic checkers which already fall under their expertise. Combining the approaches to deriving infomatics from user data [2][17] and thus automating assessment [10][9][16] from it would be the resulting step.

## 3 FORMAL REPRESENTATION

One of the primary contributions of the paper is the use of formal representation to store the drafted figures which in turn facilitates interactivity and extensibility of the tool. Before we get into the formal representation, here is a brief description of some important terms that are used throughout the paper.

- **Domain** - It refers to a given subject or the area of interest.
- **Components** - The different shapes which belong to a given domain and have some meaning in the domain. For example, states in an automata or gates in a logic circuit.
- **Lines** - Different components are connected to each other using lines.
- **Connectivity** - The connectivity between components is governed by the lines. Two components are said to be connected if a line is present between them. A component is said to connected to itself if a line references the same component.

Note - Every single component and line have various other properties such as label assigned, value, positional co-ordinates etc.

The formal notation of a given figure is defined as follows - A given figure $F$ can be represented as a three-valued tuple

$$F = (C, L, \delta) \tag{1}$$

where $C$ is the component set, $L$ is the line set and $\delta$ represents the connectivity.

A component $c \in C$ is said to be connected to another component $c' \in C$ if there exists a line $l \in L$ between $c$ and $c'$.

The connectivity $\delta$ is defined as $\delta : C \times L \longrightarrow C$ such that if $c$ and $c'$ are connected by a line $l$ then $\delta(c, l) = c'$. Similarly, the representation for a component connected to itself would be $\delta(c, l) = c$. This is internally stored as a $| C | \times | C |$ matrix where the cells in the matrix has a value $l \in L$ if there is line connecting the components.

Every single figure that is drawn on the tool is internally stored as it's formal representation. This representation is either used as it is or is transformed into a suitable representation that is domain specific. For example, this representation is used as it is in Logic Design Circuits whereas it is transformed into a transition table in the case of Finite State Automaton. The various functions which are written to operate on the figure to make it interactive, manipulate this formal representation rather than the figure itself. We have implemented the domains Logic Design Circuit, Finite State Automaton and Geometry which are very different from one another as part of our proof of concept (discussed in Section 6) to show that this formal representation generalizes to several domains which can be extremely diverse in nature.

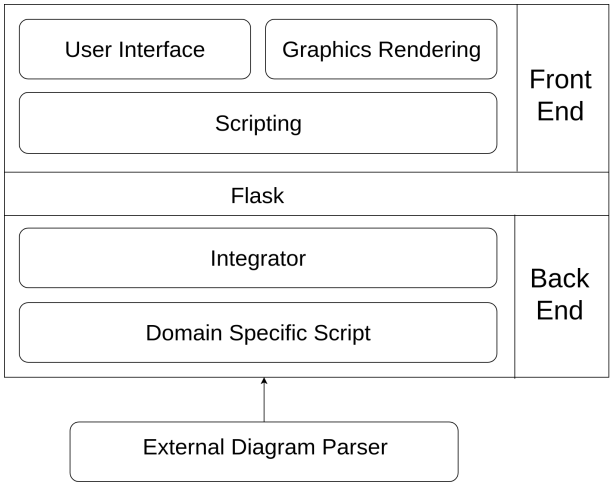


**Figure 2: High-level architectural design of the tool**

## 4 PLAYIT - OVERALL DESIGN

The overall design of our tool is broadly comprised of four components; the user interface, the scripting that supports the user interface, a server, and the back-end comprising of domain specific structures. This modularity and loose coupling was maintained throughout the high-level and low-level design such that any component can be plugged into another software environment and offer its core functionality coherently. The high-level architectural design of the same can be seen in Figure 2.

### 4.1 User Interface Design

Initially inspired by Microsoft Paint, the UI mimics the intuitive template of a white drawing canvas and its supporting toolbars. The toolbar in the header supports standard options such as using the line tool, the eraser, object selectors, color palletes and textboxes. As the back-end server supports sessioning, we allow signing in and signing out along with dedicated storage for each user. A tab menu on the left container holds our four tabs - General File Options, Domain Shapes, Domain Functions, and Shape Configuration. The file options support standard operations such as opening, updating and saving in multiple graphic formats. Depending on the chosen domain, the domain shapes hold relevant figures and drawable shapes that are displayed for the user to drag and drop onto the canvas and build their own complex structures. Figure 3 shows the various domain shapes that are a part of logic design. The domain functions display all the executable functions that can be run on the final structure present on the canvas. The shape configuring

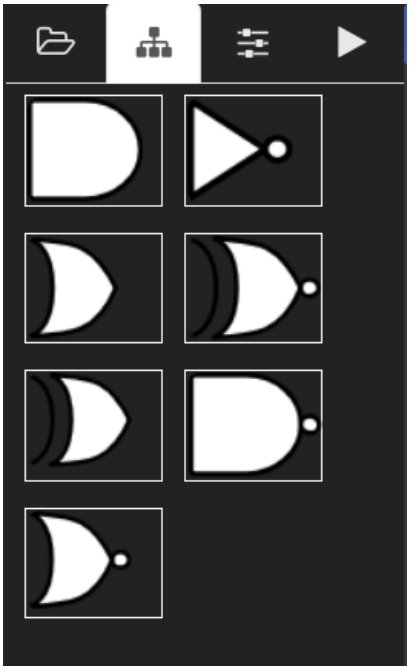

**Figure 3: The domain shapes tab for logic design**

tab contains the updatable configurations for each shape on the canvas, allowing a user to experiment with the structure's attributes dynamically. Figure 4 shows the file options and configurations tab. Apart from the tab menu, the right container holds the main canvas which renders all graphics and is resizable upto seventy percent for responsive browser sizing.

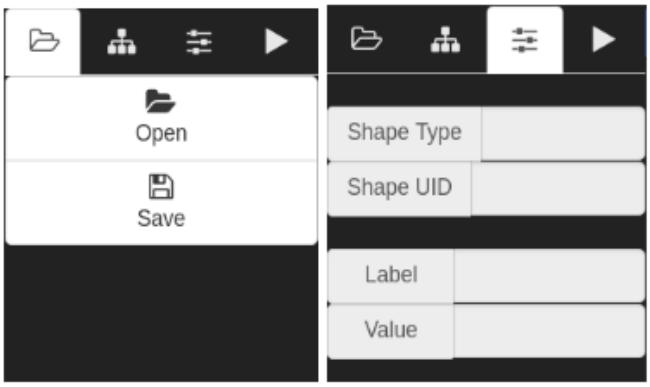


**Figure 4: The file options and configurations tabs.**

The UI design is aimed to be as minimalistic and user-friendly along with providing extensive functionalities. Following the Model-View-Controller (MVC) architectural pattern for designing the whole front-end display, we use basic HTML5, Bootstrap and AngularJS 2.0. Bootstrap helps cut-down on styling specifics and provides additional javascript extensions. AngularJS incorporates the

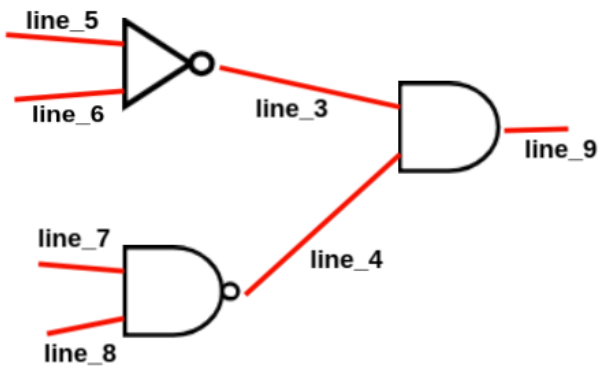


**Figure 5: An example logic design circuit drawn on the tool.**

Model-View-View-Controller (MVVM) model into the user interface and mainly supports the high dynamic nature of the canvas and user activity. The event handlers are mainly written in Javascript and JQuery. Event handlers for drag and drop, toolbar selections and graphic related functions are scripted in Javascript and talked about in the next section. Our choice of technologies was based on checking support for all browsers and changing versions, and having fallback patterns to support all internet speeds.

## 4.2 Graphic Rendering

Many popular libraries exist that offer state-of-the-art rendering functionalities and our tool works with Fabric.JS[20]. On launching the canvas, using the drag and drop handles for each shape, users can interact with the canvas and the shapes can be resized and moved anywhere on the canvas. Fabric also offers coding in custom shapes such as straight lines and curved lines which are needed to add connectivity between shapes to create complex group structures. We also handle the movement of lines [14] as we move the objects in the canvas, thus keeping connectivity intact within the shapes rendered. All coordinates are relative, thus allowing responsive UI layouts. As the user sets the configurations for each shape on the canvas, the Javascript event handlers use temporary tables that store the state of each shape and the whole diagram, along with custom properties for each diagram. Figure 5 is an example of a logic design circuit that was drawn on EasyDraw. Fabric allows converting the whole canvas into a JSON representation which holds information about each component rendered on the canvas along with the tables we use to track rendered shapes. These JSON strings can be used to save and re-render the whole composition and are the basis of our open and save file operations. These JSON representational formats also aid in easy exchange of data with the python back-end. They can also easily be converted into objects, thus aiding in processing [4].

## 4.3 Server Scripting

Since most of our core functionalities are in graphic rendering and back-end processing, our server remains simple. We chose Flask, a python micro-framework to build the server. Since the server

is minimal in functionality and does not require heavy security and standardized structures, we chose Flask instead of Django. Our server is simply a RestAPI webservice which receives calls from Angular in the front-end and sends them to the Python back-end for domain-specific processing, fetches back the response and sends it back to Angular. The main flow of the server begins by hosting the whole UI on start up. When a user logs in, a Flask session is maintained, which captures the domain chosen, previous work history, saved files and reloads all custom settings. When the user chooses a particular domain, a web request is triggered to fetch all related code-files (back-end scripts, library files, expert functions, and domain-specific shapes), and is loaded into the user interface. As the back-end functionalities are bound to the URLs generated by the server, the UI(angular) can call these API methods as get/post calls from triggered UI event handlers. When the user wants to save their work, or run an evaluation on it, the encoded JSON data prepared by the user interface scripts are sent to the server. The processing requests and responses are exchanged using JSON. We keep this JSON communication as a standard as it helps the server communicate with external software tools (discussed below in Section 5.3: Adding diagrams from another tool under Results). Exception handling and debugging are implicit functions offered in flask and Angular. Design patterns for periodic fetching to aid network traffic in multi-session environments and server-side events are implemented implicitly by the frameworks and are incorporated in our server design.

## 4.4 Back-end Domain Design

The back-end's core functionality is to use the structural composition of the graphics rendered in the front end and convert it into a collection of objects on which functions can be executed. The entire implementation of the back-end, which is coded in python, can be broadly divided into two categories, a generic interface and the domain specific structures. The structural composition and flow for a domain in the back-end can be better understood from Figure 6. For proof of concept we have developed the domains logic design, finite state automaton and basic geometry.

**Integration with the front-end -** A configuration file having the details of the functions offered by a domain along with a prompt for what the function does is used by the tool to make the domain specific interactions available to the users. An integrator module is used to link the back-end to the front-end of the tool. The integrator is responsible for creating the domain specific objects and calling the respective functions. The server passes the function name and arguments list along with the domain object to the integrator which then calls the function and returns the result to the server in the form of a JSON string. Thus the back-end communicates only with the Flask server, and so can be plugged into other coding environments as a fully functional library on its own.

**Basic Interface -** A basic interface class provides the base implementation that any domain must implement. A constructor function of the basic interface implements the creation of the figure object which comprises of the components, lines and the connectivity details. The UI scripting component of the tool is responsible

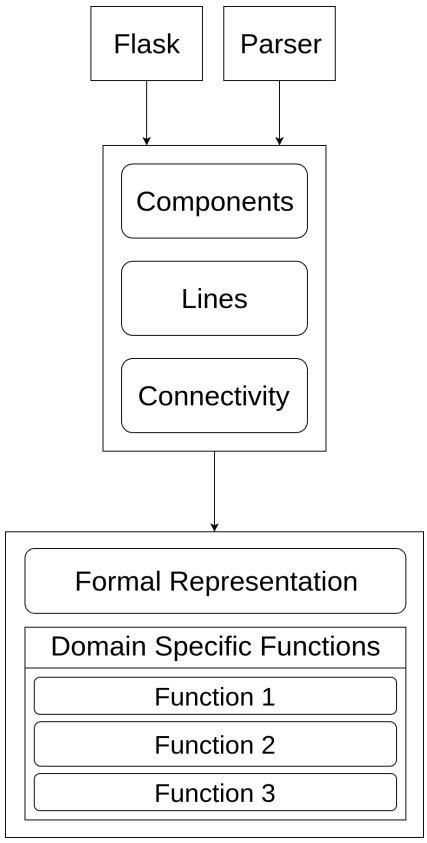


**Figure 6: Back-end design: The structural composition and flow for a domain in the back-end**

for providing all the necessary details that need to be stored by the back-end. These details are used for creating the necessary elements that are responsible for representing a figure (type of the shape, coordinates of the shape on the canvas, the rotational angle, labels and associated configured user inputs). A component class is used as the base interface that each component must implement. The constructor function of the component class initializes all the attributes for each component as provided by the user interface. A line class is used as the base interface that each line in the figure must comprise. A constructor function of the line class initializes all the attributes for each line as provided by the user interface. The connectivity is given in the form of an adjacency matrix and denotes the relationships between the various components and the lines that link two components. There are additional utilities provided to save the figure, reload a figure and for parsing the JSON being exchanged.

**Domain specific details -** The domain class inherits the basic interface class and builds on it. The constructor function initializes the figure object whose properties include the components, the lines and the connectivity. The components are objects of the component class. The lines are objects of the lines class. The connectivity is represented in the form of an adjacency matrix providing details about the connected components and the line that connects two components as the default implementation. However, this can

be modified to suit the domain of interest. For example, for finite state automaton, a transition table can be used as it provides details about the states, the input alphabets and the corresponding state it transitions to for a given input alphabet. In domains like geometry, where there is no implicit graph like structure, the connectivity information is ignored and only the lines information is manipulated. Functions can be added to interact with this figure object and the results are returned to the server as JSON strings.

**XML parser -** As discussed below in section 5.3, Adding diagrams from another tool under Results, an explicit parser needs to be built to convert the diagrams representation, we used the XML representation from JFLAP [12], into the format that the tool expects. This includes information about the components, lines and the connectivity between the various components. We make use of the XML.etree.ElementTree module in python to parse the XML representation and then create the structures necessary for the tool to render the diagram. They are then transformed into the representation that is suitable for all back-end operations.

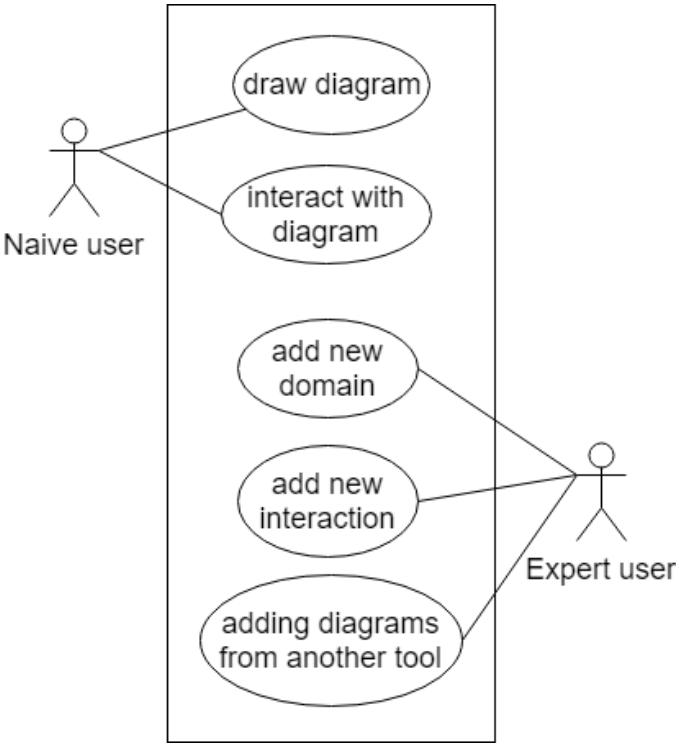


**Figure 7: Use Case Diagram: The various possible interactions of an expert and a naive user with the tool**

## 5 EXTENSIBILITY

One of the primary goals was the extensibilty of the tool. Figure 7 shows the various interactions the users can have with the tool. It also shows the different ways in which the expert user can extend the tool by adding various domains and interactive functions. To demonstrate the simplicity in extension of the tool, we provide the high-level algorithms explaining all the steps required for an expert user to add the necessary features.

### 5.1 Adding a new domain to the tool

To add a new domain to the tool, we first need to add all the necessary images that are meant to be a part of the domain in a specific folder. The user is also required to make a configuration file stating all the functions the domain provides. An integrator file is responsible for all the function calls that are made to the back-end. The user is required to append to this file to ensure that the arguments are parsed correctly and the function call is made. The returned value is then sent back to the UI via the Flask interface.

As mentioned earlier, the loose coupling of the UI and back-end helps maintain the same user experience, irrespective of what domain is being added. The new shapes are added dynamically into the shapes tab, and the new functions into the execution tab, thus maintaining the same flow of activity for all the users.

Following this, the user now has to handle saving and reloading of the diagrams. The UI passes three forms of tables - the components, the lines, and the connectivity between the components. The user is responsible to parse these tables into a structure suitable for the formal representation of diagrams in the domain. For example, a transition table is maintained for representing a finite automaton. Once the user has this representation defined, they can perform necessary interactions with it. They can have a set of predefined functionality that can now be accessed by the UI. Note that there can be additional functions added later on as well, as explained in the section below. We have demonstrated this by implementing the domains digital logic circuit, finite state automaton and basic geometry.

### 5.2 Adding new interactions to an existing domain

As mentioned above, any changes made to the domain with respect to the interface needs to be reflected in the configuration file and the integrator file to ensure that the function is being called correctly. Apart from this, the user has complete freedom to manipulate the given representation and define any interaction on it. Once defined, the function now becomes available on the UI and can be accessed by other users. We have demonstrated this by having multiple functions in each domain. For example, checking if the given automaton is a deterministic finite state automata or if the given string belongs to the language. We have functions like evaluating the circuit and generation of truth table for digital logic circuits.

### 5.3 Adding diagrams from another tool

In order to load the representation of a diagram (like XML) from other tools, the user is required to build a parser that converts the given representation into a form that is acceptable by the tool. This includes the three structures having information about the components, lines and connectivity. This can now be passed to an existing domain so it is transformed into the desired representation or the user can add a new domain to the tool by following the steps mentioned above. Similarly, the existing functions can be called or new functions can be added. The parsed results can be sent back to the UI to render the diagram on the canvas. To conclude, once the

representation (from a different tool) is parsed, it can be loaded on to our tool and various interactions can be performed with it.

We have successfully demonstrated this by carrying out the stated procedure on JFLAP [12]. A parser was built to parse the XML representation of diagrams that were originally drawn on JFLAP and the results were passed to the domain-specific script corresponding to the finite state automaton domain in the back-end. The diagram can be rendered back to the UI and the defined interactions can be performed on the loaded diagram.

### 5.4 Adding functions for automated evaluation

Additional functions can be added to the tool by a teacher to formulate questions using the same procedure as above. For a given question, a student user may input their response which can then be verified by the tool automatically. Instead of returning the result of a given interaction directly, the teacher can check if the result matches the students response and automatically evaluate them.

## 6 RESULTS

Keeping the existing work in mind, we were able to develop PlayIt, a tool with an extensible architecture for adding new domains with relevant functions using the proposed solution. We also allow users to load diagrams that were drawn on other platforms and allow them to perform the necessary interations with them. We were successfully able to design the tool while ensuring none of the back-end is coupled with the tool. This allowed us to easily add and remove domains by just making changes to the integrator module and adding domain specific back-end script. Figure 8 demonstrates the whole UI along with an example finite state automaton that accepts even binary strings. You can also view the various ways in which the user can interact with the diagram on the execution tab on the left side of the canvas.

As a proof of concept, we have successfully extended the tool with three domains, which are, digital logic circuit design, finite state automaton and basic geometry. We have also added various functionality to these domains allowing users to interact with the diagrams. The domain specific interactions that are added currently for the respective domains are as follows:

### Digital Logic Circuit Design

- Evaluation of the circuit, given inputs.
- Generation of truth table for the given circuit.
- Generation of random inputs for the circuit (useful in examinations where the student has to specify the expected output by analyzing the behaviour of the circuit for a given input).
- Automatic evaluation of user's output after evaluating the circuit based on generated input.

Figure 9 shows the execution tab which comprises of all the available interactions for the digital logic circuit design domain.

### Finite State Automaton

- Check if the given input string is a part of the language represented by the finite state machine.
- Generates the transition table for the given finite state machine (the transition table is generated for both deterministic finite automaton as well as non-deterministic finite automaton).
- Generation of random inputs string using the alphabets of the language.
- Automatic evaluation of user's output to whether the given string belongs to the language.
- Checks if the nature of the given automaton is deterministic or non-deterministic.

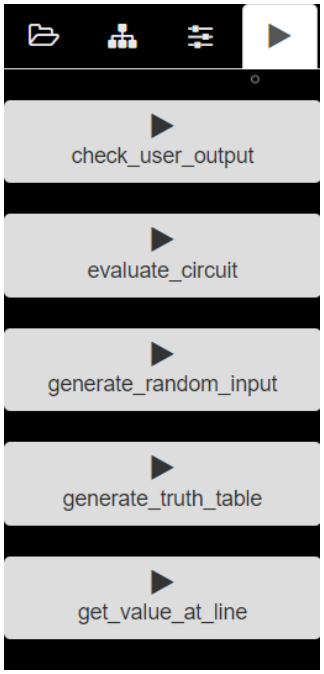


**Figure 9: The execution tab with all interactions for Digital Logic Circuit Design.**

### Basic Geometry

- Calculate the perimeter and area of the given triangle.
- Calculate the circumcenter of the triangle.
- Calculate the centroid of the triangle.
- Calculate the incenter of the triangle.
- Calculate the orthocenter of the triangle.

Note that these are just some sample functions designed to demonstrate proof of concept. It is not an exhaustive list of all the interactions that can be added to a domain.

## 7 EVALUATION

Previous studies in computer-aided education [13][10][9][17][5] have suggested that the availability of custom and familiar tool development environments helps in production of more intuitive and precisely defined solutions and problems. As mentioned earlier, our previous survey suggested that extending and integrating new module components was perceived by many users, instructors and students, as the lacking feature in most training tools. Tools,

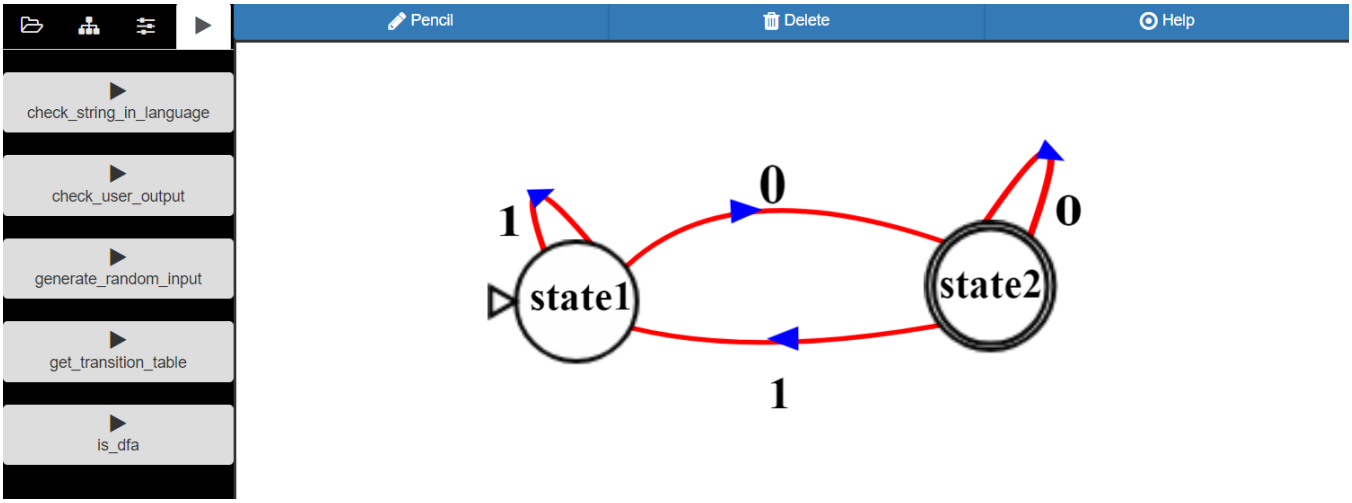


**Figure 8: Basic design of the UI. A DFA drawn onto the canvas.**

like Adobe and COLDVL, that did offer an adaptation interface either required a steep learning curve or allowed rudimentary additions. Most of the time, instructors are asked to follow certain pre-defined steps and procedures to utilize all the features of the tool. Any creativity they might have is consequently limited, especially with regard to producing and designing varied problem statements for the students. The role of graphical tools, in these scenarios, is not questioned by the teachers, and is looked as only a range of drawing techniques. The implicit methods supported impose choices on both type of users and thus do not offer an optimal method to create relevant contexts. In light of the amount of time teaching-learning processes include usage of computer-aided designing, we found it key to offer the intuitive freedom of a designer to the users directly. Our user studies reflect experiences of teachers and pupils using this tool in their development framework and in modelling assignments of their respective training subjects.

We conducted a short presentation which was followed by an anonymised survey to collect feedback and comments. Below we summarize the responses we received from the several professors and students belonging to varying majors in different years of their undergraduate studies.

**Do such interactive visualizations actually help teaching - learning process?** Most participants appreciated the freedom of designing a system and being able to control it's functionality dynamically. This seemed to enhance their learning experience by allowing them to observe the outcomes of mistakes in detail and helping them precisely pick out the solutions. Two instructors mentioned that using the tool to show sequential design patterns in Finite Automata and the incremental outcomes, helped their students understand the concept faster than normal, and even encouraged students to experiment with their own design and patterns. A short survey among students showed that majority of the students preferred the courses to be more interactive and agreed that the visualizations and graphic tool made the coursework easier to grasp. A majority of the professors agreed that having visualizations that they could interact with helped present subject matters using narratives, which students followed better than blocked learning.

**Were you able to represent visuals of a conventionally non-graphic subject?** Seeing how easy it was to flood in visual data and interactively create multiple drafts of these visual elements, a few participants confirmed the extensibility of the tool in pure graphics as well. Subjects such as Data Structures, Advanced Algorithm Design, were adapted and tested, by importing basic 2D shapes and navigation elements, helped in converting complex data structure designs into easily explainable visualizations. Three students managed to represent the reversal and functioning of a Linked List, which was "very helpful since we could show how the design was implemented in code simultaneously".

**When would you choose this tool over existing solutions?** Several participants agreed that being able to write up and add in their own functions, related to changing subject requirements was a fresh feature that the tools that they were using previously lacked. Few professors expressed their interest in using the tool for automatic evaluation of assignments and exams. Since each graphic draft is dumped into a tangible code structure, an automatic checker would be able to run through the structure and match validation checks set by the instructors. Several instructors also expressed that such a tool would be able to check for plagiarism to ensure that students did not copy the solutions. They liked being able to have multiple domains in the same tool as it saved time and retained familiarity in usage. Several of them also expressed their views on lack of existing tools that allowed them to interact with the diagrams from different domains. They would have to explore multiple tools which specialize in a particular domain, to find the most suitable for their use case.

**How did you like interacting with the tool?** Participants who used it purely from a users or students perspective found the tool

simple and intuitive. They seemed to find drawing drafts and using general functionalities similar to existing tools, although all agreed for general improvements in the overall polishing of the look and feel of the tool. Professors and Instructors who were able to add in their own modules of custom graphic elements and functionalities appreciated the simplicity of the interface template. For participants who were of non-coding backgrounds, a short introduction was held to help them add in their use-cases. They later expressed that they would be able to continue development independently due to the ease of coding in Python. Students and Teaching Assistants both mentioned that they were able to use the tool to make their own practice assessments and teaching courses, switching roles as instructors to understand problems better.

**Do you have any concerns regarding the tool?** One of the primary concerns expressed by professors who of non computer science backgrounds was the lack of programming experience. But upon explaining the backend design and showing a demo of how a new domain can be added to the tool with just the basic knowledge of python, these concerns were no longer present. Few participants questioned if the tool would support using other programming languages apart from Python to write-up their modules and add it into the tool. This was cleared when we explained the template behaving like a generic API and thus virtually supporting any code base of any preferred coding language.

**Any additional comments** Few professors expressed their interest in using such a other domains that they are interest in incorporating into the tool which varied from electrical engineering, databases and even differential mathematics. Few students also volunteered to help the teachers in extending the tool to other domains.

## 8 APPLICATIONS

One of the most prominent applications for a pluggable web-based drawing tool such as this is in the domain of education. With online school and computer-based exams becoming more and more common among institutions around the world, this tool could be a great way to help students incorporate diagrams to support their answers. It also helps teachers by automating the evaluation of such answers. Since the diagrams are internally stored as formal representations rather than images, it becomes easier to compare them and evaluate them without the need for complex image processing tasks. This makes automated evaluation robust and reliable. The pluggable nature of the tool allows any user who is an expert in a given domain to add to the tool. Students no longer have to get familiar with different software for various subjects and can use this as an all-in-one tool serving all their needs for drawing technical diagrams.

## 9 CONCLUSION AND FUTURE WORK

We have successfully developed PlayIt, a pluggable web-based drawing tool, and tested it by plugging in the domains like logic design, finite state automaton and basic geometry, each having it's own formal representation for the diagrams along with multiple interactions that can be performed on them. This is achieved without making any changes to the tool itself. The flexibility and freedom this tool provides the users can be immensely exploited to create custom functions as per one's needs and requirements.

In the future, the tool can be extended to several other domains of the users choice and can be open sourced for other people to contribute. Image processing can be used to interpret existing images into the formal representation so different interactions can be performed on them. Given a parsing tool that can convert any image into a representation that the tool can interpret, we can design a web plugin to automatically make all diagrams on websites like Wikipedia interactive so students can not only read about a topic but can also interact with the figures to understand concepts better.

Currently, expert users are required to write the domain specific script; however, we are looking at ways in which this can be automated as well. Given the formal representation and set of rules describing various functions, we want to design a code generator that will automatically generate the domain specific script. We also hope to incorporate this in the computer-based exams that are conducted in the university and use it as a teaching tool in the classes.

## REFERENCES


[1] Adobe.com. 2019. Adobe Illustrator. Retrieved Jan 31, 2019 from https://www.adobe.com/in/products/illustrator.html

[2] Umair Z Ahmed, Pawan Kumar, Amey Karkare, Purushottam Kar, and Sumit Gulwani. 2018. Compilation error repair: for the student programs, from the student programs. In *2018 IEEE/ACM 40th International Conference on Software Engineering: Software Engineering Education and Training (ICSE-SEET)*. IEEE, 78–87.

[3] AutomataSimulator.com. 2019. Automata Simulator. Retrieved Jan 31, 2019 from http://automatonsimulator.com/

[4] Tavmjong Bah. 2011. *Inkscape: guide to a vector drawing program*. Vol. 559. Prentice Hall Boston.

[5] David Baneres, Robert Clarisó, Josep Jorba, and Montse Serra. 2014. Experiences in digital circuit design courses: A self-study platform for learning support. *IEEE Transactions on Learning Technologies* 7, 4 (2014), 360–374.

[6] Fulvio Corno, Luigi De Russis, and Juan Pablo Sáenz. 2018. Easing IoT development for novice programmers through code recipes. In *2018 IEEE/ACM 40th International Conference on Software Engineering: Software Engineering Education and Training (ICSE-SEET)*. IEEE, 13–16.

[7] Draw.io. 2019. Draw.IO Homepage. Retrieved Jan 31, 2019 from https://www.draw.io/

[8] Jon Ferraiolo, Fujisawa Jun, and Dean Jackson. 2000. *Scalable vector graphics (SVG) 1.0 specification*. iuniverse.

[9] K Nee Goh and R Hilisebua Manao. 2013. Assessing engineering drawings through automated assessment: discussing mechanism to award marks. *International Journal of Smart Home* 7, 4 (2013), 327–335.

[10] Kim Nee Goh, Siti Rohkmah Mohd Shukri, and Rofans Beleam Hilisebua Manao. 2013. Automatic assessment for engineering drawing. In *International Visual Informatics Conference*. Springer, 497–507.

[11] Saul Greenberg, Mark Roseman, David Webster, and Ralph Bohnet. 1992. Issues and experiences designing and implementing two group drawing tools. In *Proceedings of the Twenty-Fifth Hawaii International Conference on System Sciences*, Vol. 4. IEEE, 139–150.

[12] JFLAP.org. 2019. JFLAP. Retrieved Jan 31, 2019 from hhttp://www.jflap.org/

[13] Patrice Laisney and Pascale Brandt-Pomares. 2015. Role of graphics tools in the learning design process. *International journal of technology and design education* 25, 1 (2015), 109–119.

[14] Juan Pineda. 1988. A parallel algorithm for polygon rasterization. In *ACM SIGGRAPH Computer Graphics*, Vol. 22. ACM, 17–20.

[15] Antoine Quint. 2003. Scalable vector graphics. *IEEE MultiMedia* 10, 3 (2003), 99–102.

[16] Gargi Roy, Devleena Ghosh, Chittaranjan Mandal, and Indraneel Mitra. 2015. Aiding teaching of logic design and computer organization through dynamic problem generation and automatic checker using COLDVL tool. In *2015 IEEE Seventh International Conference on Technology for Education (T4E)*. IEEE, 15–22.

[17] Zarko Stanisavljevic, Bosko Nikolic, and Jovan Djordjevic. 2012. A module for automatic assessment and verification of students' work in digital logic design. In *2012 IEEE 19th International Conference and Workshops on Engineering of Computer-Based Systems*. IEEE, 275–282.

[18] Renata Vaderna, Željko Vuković, Igor Dejanović, and Gordana Milosavljević. 2018. Graph Drawing and Analysis Library and Its Domain-Specific Language

for Graphs' Layout Specifications. *Scientific Programming* 2018 (2018).

[19] Vectr.com. 2019. Vectr Drawing Tool. Retrieved Jan 31, 2019 from https://vectr.com

[20] Mary C Whitton. 1984. Memory design for raster graphics displays. *IEEE Computer Graphics and Applications* 4, 3 (1984), 48–65.