frontiers | Frontiers in Physics

TYPE Original Research
PUBLISHED 12 October 2022
DOI 10.3389/fphy.2022.944086

OPEN ACCESS

EDITED BY
William Frere Lawless,
Paine College, United States

REVIEWED BY
Eugene Santos,
Dartmouth College, United States
Jean-Sébastien Sottet,
Luxembourg Institute of Science and Technology (LIST), Luxembourg

*CORRESPONDENCE
Nicolas Hili,
nicolas.hili@univ-grenoble-alpes.fr

SPECIALTY SECTION
This article was submitted to Interdisciplinary Physics, a section of the journal Frontiers in Physics

RECEIVED 14 May 2022
ACCEPTED 22 September 2022
PUBLISHED 12 October 2022

CITATION
Castellanos-Paez S, Hili N, Albore A and Pérez-Sanagustín M (2022), BOARD-AI: A goal-aware modeling interface for systems engineering, combining machine learning and plan recognition. *Front. Phys.* 10:944086. doi: 10.3389/fphy.2022.944086

BOARD-AI: A goal-aware modeling interface for systems engineering, combining machine learning and plan recognition

Sandra Castellanos-Paez[1], Nicolas Hili[1]*, Alexandre Albore[2] and Mar Pérez-Sanagustín[3]

[1]Université Grenoble Alpes, CNRS, LIG, Grenoble, France, [2]Onera DTIS, Toulouse, France, [3]IRIT, Université de Toulouse, CNRS, INP, UT3, Toulouse, France

Paper and pens remain the most commonly used tools by systems engineers to capture system models. They improve productivity and foster collaboration and creativity as the users do not need to conform to formal notations commonly present in Computer-Aided Systems Engineering (CASE) tools for system modeling. However, digitizing models sketched on a whiteboard into CASE tools remains a difficult and error-prone activity that requires the knowledge of tool experts. Over the past decade, switching from symbolic reasoning to machine learning has been the natural choice in many domains to improve the performance of software applications. The field of natural sketching and online recognition is no exception to the rule and most of the existing sketch recognizers rely on pre-trained sets of symbols to increase the confidence in the outcome of the recognizers. However, that performance improvement comes at the cost of trust. The lack of trust directly stems from the lack of explainability of the outcomes of the neural networks, which hinders its acceptance by systems engineering teams. A solution shall not only combine the performance and robustness but shall also earn unreserved support and trust from human users. While most of the works in the literature tip the scale in favor of performance, there is a need to better include studies on human perception into the equation to restore balance. This study presents an approach and a Human-machine interface for natural sketching that allows engineers to capture system models using interactive whiteboards. The approach combines techniques from symbolic AI and machine learning to improve performance while not compromising explainability. The key concept of the approach is to use a trained neural network to separate, upstream from the global recognition process, handwritten text from geometrical symbols, and to use the suitable technique (OCR or automated planning) to recognize text and symbols individually. Key advantages of the approach are that it does not resort to any other interaction modalities (e.g., virtual keyboards) to annotate model elements with textual properties and that the explainability of the outcomes of the modeling assistant is preserved. A user experiment validates the usability of the interface.

KEYWORDS

machine learning, plan recognition, artificial intelligence, systems engineering, sketch recognition, model-driven development

1 Introduction

Despite its proven modeling value in many engineering domains, Computer-Aided Systems Engineering (CASE) tools have only had moderate acceptance by system engineers and architects to assist them in their day-to-day tasks [1]. The complexity of creating, editing, and annotating models of system engineering takes its root from different sources: unsuitable representations, outdated interfaces, laborious modifications, and difficult collaborations [2].

As a result, especially in early development phases, systems architects tend to favor more traditional tools, such as whiteboards, paper, and pencils over CASE tools to quickly and easily sketch a problem and its solution. Among the benefits of sticking to traditional tools, whiteboards foster collaboration and creativity as the users do not need to strictly conform to formal notations.

A common pitfall for using traditional tools, however, is that human users are required to reproduce any sketched solutions inside of formal tools when it comes to formalizing the models. Modern post-WIMP[1] interfaces (e.g., electronic whiteboards) could help to automate this task by allowing users working on a digital representation of the model, one that can be directly exported, to be modified *via* modeling tools. Bridging the informality of the working sketches captured on interactive whiteboards with formal notations and representations has the potential to lower the barrier of acceptance of CASE tools by industry [3, 4]. This acceptance can be obtained by automatically or semi-automatically translating informal sketches into their corresponding formal elements using a specific and conventional notation.

Natural sketch recognition aims at bridging the gap between free-form modeling and formal representations using dedicated graphical notations. A significant body of related work can be found in the literature, spanning offline and online recognition [5–13]. Offline recognition allows users to capture sketches using pens and paper and to further digitizes them, while online recognition relies on interactive digital displays such as electronic whiteboards for user inputs [6]. With advances in modern post-WIMP interfaces, recent pieces of work tend to favor online sketch recognition systems over offline ones. Yet, providing a robust online sketch recognition is still a hot research topic and is not well-settled in systems engineering.

1.1 An early model recognition assistant

In our previous work [14, 15], we suggest the use of symbolic Artificial Intelligence (AI) techniques to aid systems engineers to design models in a freehand way using large multi-touch screens. The heart of this approach lies in the use of goal recognition techniques to translate user's sketches into model elements. More precisely, a modeling assistant identifies the most probable model elements intended to be drawn by a user from an initial sketch, even when partial. The outcome of the assistant is a list of suggestions ordered by the probability that a complete model element corresponds to the user's intent. This probability is based on the "distance" between the partial sketch and any possible model elements that can be drawn from that partial sketch measured in terms of the number of steps that would remain to finish drawing the model element completely.

The main benefit of relying on symbolic AI rather than on Machine Learning (ML) is explainability. "Explainability" is the property of a system that provides an output that makes understandable to the human user the reasons of an algorithm's choice. This is a condition needed by any process-directed tool that allows users to evaluate the criteria behind a choice to use the tool more efficiently [16]. Not only the modeling assistant provides the user with a list of suggestions, but it also details the remaining steps to draw the suggested model elements completely. Our preliminary evaluation suggests that recognizing complex shapes (e.g., an operational actor made of four straight lines and one circle) using AI methods is suitable for online incremental recognition. In the present study, we make the following contributions:

1) We refine a part of the approach that no longer relies on goal recognition alone, but rather on the combination of symbolic AI and ML techniques. Handwritten text and geometrical shapes composing model sketches require two distinct recognition processing and, thus, must be decoupled prior to the recognition process to occur. We train a Neural Network (NN) to distinguish text from geometrical shapes such that handwritten text can be recognized using traditional Optical Character Recognition (OCR) engines while geometrical shapes are determined by our initial goal recognition algorithm to identify the model elements. This approach allows us to enhance our recognition process to identify model elements annotated with text without resorting to virtual keyboards or voice recognition. We present the extended approach, describe a training platform we developed to train the NN, and summarize the results of the training process.

1 Windows, mouse, and pointer interfaces.

2) We reformulate the representation of the sketching environment used by our shape recognition engine in order to improve the speed and accuracy of the goal recognition process, and to make it more tolerant to drawing imprecision. This new representation, expressed in the PDDL language proper to automated planners, is lighter than the one previously used, hence speeding up the recognition process. On the one hand, the simplification of the planning representation comes at the cost of more ambiguity when recognizing model elements from (very) partial drawings. On the other hand, it led to better results in real case scenarios, as completing a sketch adds new constraints, thus removing most of the ambiguity. In addition to these improvements, we replace the previous search algorithm by the anytime algorithm used in LAMA [17]. It results in a generation of plans which is faster, and that in time provides plans with increasing quality.
3) The refined approach and enhanced recognition algorithm resulted in a modeling environment called BOARD-AI. It consists in an electronic whiteboard interface coupled with both shape and text recognition software, and an automated planning algorithm that provides completion suggestions for the sketch drawn by the users.
4) Finally, this study presents an early evaluation of the human-machine interactions and of the usability of BOARD-AI on two groups of users. These users employed the modeling environment to design a system engineering system before answering to a questionnaire that we then evaluated. This study provides a first assessment of the validity of our approach, and of the trust that users are willing to place in an artificial intelligence-based recommendation system. This evaluation protocol eventually provided us with useful indications on future improvements of the modeling framework.

1.2 Paper structure

The rest of the study is structured as follows: Section 2 presents background concepts; Section 3 presents the approach and describes the implementation of BOARD-AI; Section 4 describes the user evaluation of BOARD-AI and the conclusions we drew; Section 5 presents related work; and Section 6 concludes.

2 Background

The term sketch has a very broad definition. It includes a variety of freehand drawings made by amateurs or professionals, such as doodles, clip-arts, caricatures [13], but also drawings used in various engineering domains, e.g., electrical circuits. Natural sketching is becoming more and more popular due to the increased use of post-WIMP interfaces, including interactive whiteboards, interactive walls, large multi-touch screens, interactive pen displays, and personal tablets [9]. The emergence of AI-enabled sketching tools such as Google Autodraw also largely contributed to the digital revolution of natural sketching from a very broad range of applications and use cases.

The present study targets sketches used to capture models of systems in a more natural way using traditional software and systems engineering languages. Sketching tools offer an alternative approach to traditional CASE tools to rapidly design a model where users have more freedom and flexibility as they are not required to learn how to use complex tools to create the models desired [7]. However, sketching tools dedicated to modeling differ from more general-purpose sketching tools on two fundamental aspects.

First, sketches vary in terms of representation, size, and style, and general-purpose sketching tools have to handle a large number of individual sketches. As such, recent work tends to favor classification techniques and complex deep learning models to handle such variation [13], for it relies on large sets of sketches to train the recognizers. As opposed, diagrams in computer science and systems engineering rely on relatively stable and simple representations composed of simple geometrical shapes (rectangles, circles, *etc.*).

Second, text is omnipresent in traditional diagrams in computer science or systems engineering to label model elements. However, text and shapes do not rely on the same training models to be efficiently recognized. The duality of recognizing text and symbols individually has been explored in the literature, e.g., in [11] and takes its source from traditional text/non-text separation techniques in offline document analysis [18]. While some work moved the problem aside by providing users with alternative text editing capabilities (e.g. [19]), others (e.g. [11, 20], provide seamless modeling capabilities, for it relies on segmentation techniques to separate text and non-text before starting to recognize shapes and text individually. In our previous work [14, 15], we relied on alternative editing capabilities, including virtual keyboard and voice recognition to annotate model elements with text. Our present work adheres to this second approach as it seems more natural for users to only rely on a single modality to draw models.

2.1 Classification

One trend to recognize modeling elements is to rely on AI tools and algorithms, more specifically, on ML techniques based on NNs. This family of approaches typically involves two phases [21]. During the training phase, algorithms are trained to recognize elements based on pre-existing libraries. During the

recognition phase, these algorithms can identify elements with a certain degree of confidence.

NNs can learn and generalize from training data; they are particularly fault and noise tolerant [22]. Thus, they are often used in several domains for the classification of input data into categories [23–25]. An essential element of NNs is the neuron. A neuron is an information processing unit taking several inputs and producing one output [26]. Each input has its own weight; the neuron calculates the sum of the weighted inputs plus a bias term (this term represents how easily the neuron fires). Afterwards, this sum is passed trough an activation function to obtain the output. The role of an activation function is to introduce non-linearity into the output of a neuron. It also determines whether and how much a neuron should be activated, and its choice may improve or reduce the neuron's performance. Neurons connect to each other to form NNs; a neuron's output then provides the input to another neuron.

Most NNs are organized into multiple layers, and each layer has a specific number of neurons. We can distinguish three types of layers [27]: the input layer that provides data from the world to the network (in our case, the extracted features from the user's drawing); hidden layers that compute and transfer information from that input layer to the output layer; and the output layer that corresponds to the output prediction of the network. Multiple hidden layers can be stacked along each other. A network is called fully connected if every node in each layer is connected to every adjacent node in the adjacent forward layer.

NNs are widely used to classify data when a labeled dataset to train it is provided (supervised learning). A NN classifier tries to approximate a function that maps all of the elements in a space (the elements to classify) to the elements in another space (the categories of these elements). The network, by adjusting weights and biases, tries to approximate this function as best as possible, and then maps the elements to be classified to their respective categories.

In this work, we implement these concepts to develop an NN classifier able to find a function that maps collected data from user's drawings to one of two categories, either text or geometrical shape. To do so, we represent the training data as a label dataset consisting of a set of features (e.g. the number of sharp corners, a bounding box ratio, etc. See Section 3.3) and a target (the corresponding category, i.e., text or geometrical shape).

2.2 Automated deterministic planning

AI planning [28] has been used to perform activity recognition in the context of a system managed by human operators whose currently pursued operational goal has yet to be determined [29]. Several goal recognition [30] fields of application have surfaced, including "operator modeling" to improve the efficiency of man-machine systems. Early applications of the approach failed because of the complexity of plans, the issues due to evaluating actions that did not fit any plan, or the issues from interleaving planning and execution. Moreover, the work in goal recognition has historically proceeded independently from the planning community, using handcrafted libraries rather than planners [31].

An automated planning task can be represented as a directed graph model, where the nodes correspond to the different situations (or states) in which a system can be, and the edges represent actions that drive the system from one situation to a new one. Solving a planning problem consists in finding a sequence of actions $\langle a_1, \ldots, a_n \rangle$, also called plan π, that drive the system from an initial state to a desired goal, or a final situation. The length $|\pi|$ corresponds to the number of actions in the plan π: the length of a plan is commonly considered as a preference criterion to evaluate it.

To achieve automated deterministic planning, we adopt the STRIPS formalism [32]. In STRIPS, a factored representation represents states *via* a set of Boolean variables, interpreted as a conjunction, and such that each state s is a complete assignment of state variables. A planning problem is then defined as a 4–tuple $\langle \mathcal{F}, \mathcal{A}, \mathcal{I}, \mathcal{G} \rangle$, where $\mathcal{F}$ is the set of state variables (assuming Boolean values), $\mathcal{A}$ is the set of operators (or actions), and $\mathcal{I}, \mathcal{G} \subseteq \mathcal{F}^2$ are two sets of variables describing the initial state and the goal state(s), respectively. An action $a \in \mathcal{A}$ is defined as the 3–tuple $\langle$ pre(a), add(a), del(a) $\rangle$, where pre(a) is the set of preconditions of a, add(a) and del(a) are the sets of post-conditions of a, respectively defining the set of propositions added and deleted from the state. The pre-conditions determine in which state an action can be applied, while post-conditions specify the changes to variable assignments made by applying the action in a state. In other words, an action a is applicable in state s iff pre(a) $\subseteq s$, where the application of a in s is defined by the transition function T (s, a) = (s/del(a)) $\cup$ add(a).

In order to solve these planning problems, we adopt in this study an anytime approach. The planning algorithm first runs a search in the graph, aimed at finding a solution as quickly as possible. Once a plan is found, it searches for progressively better solutions by running a series of more expensive searches (in terms of computation time). The cost of the best known solution is used for pruning the subsequent searches. The final result is a set of solution, obtained at increasing time intervals but with (hopefully) a better solution quality.

Anytime algorithms are usually based on Weighted A*: an heuristic search algorithm that uses a weight to scale the heuristic value of each node of the graph about to be visited [33]. The underlying idea is to continue the search after the first obtained solution, possibly adjusting search parameters like the weight or pruning bound, and thus progressively find better solutions [34–36].

In our previous work [15], we describe an approach based on AI automated planning to recognize complex sketches

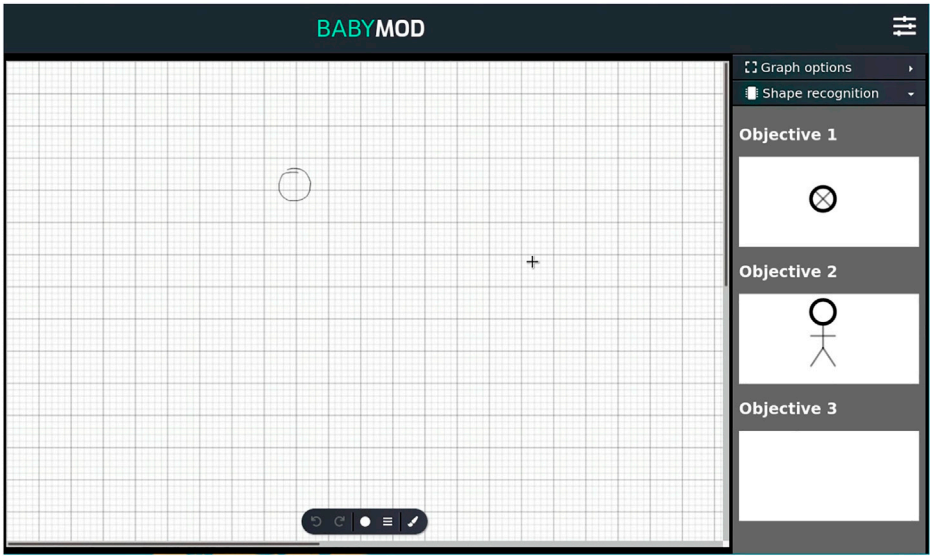

FIGURE 1
A preliminary implementation of the modeling assistant [15]. The central area is an HTML5 Canvas where the user can sketch model elements. The right sidebar shows suggestions to complete the sketch.

representing model elements and to guide users in their completion. For our preliminary sketching tool (see [15]), we adopted a strategy that uses the planning framework for goal recognition to perform the converse task of automated planning [37], i.e., recognizing the most probable goal given an initial state and a plan. Here, the task is to identify the shapes yet to be drawn and their placement to create a meaningful system-engineering sketch from an incomplete drawing. We use a goal library to describe the possible solutions of a plan. Sketching is therefore represented as a planning problem, where the initial state corresponds to a partial drawing from the user and the goals represent the different model elements the planner is able to recognize. The actions represent the different operations a user or a hypothetical drawing-agent would perform to complete an initial sketch.

3 Materials and methods

Figure 1 is an overview of a preliminary implementation of the modeling assistant. As soon as the user starts to draw some shapes on the screen, the modeling assistant is able to propose suggestions in terms of systems engineering sketched elements. Suggestions are ordered based on the length of the plan π calculated by the planer. Explainability is a central concern in our implementation: suggestions of the final shape form—calculated by the modeling assistant—is in direct correlation with how much of the complete model element sketch is left to draw.

One limitation of our initial implementation is that it does not support handwritten text annotating model elements. Automated planning is usually less fault-and-noise tolerant than other techniques such as NNs. It is therefore not suitable for text recognition, traditionally done through OCR. For our preliminary sketching tool, we then relied on two other interaction modalities. A user can annotate a model element through a virtual keyboard, or through voice recognition. This implies that the model element is first recognized before it can be annotated.

In this study, we extend our initial approach with handwritten text annotation support. The key concept is to use a trained neural network upstream from the global recognition process to separate handwritten text annotation from geometrical shapes, and to use the suitable technique (OCR or automated planning) to recognize text and shapes individually.

3.1 Approach definition

An overview of the extended approach is illustrated in Figure 2. The sketching work starts with a user's drawing. That drawing can be partial (i.e., it only represents a part of an element to be recognized) or complete (it completely represents the element to recognize). Compared to our previous approach, the drawing can be annotated with handwritten text. We assume in our approach that text and geometrical shapes belong to two different classes of problems

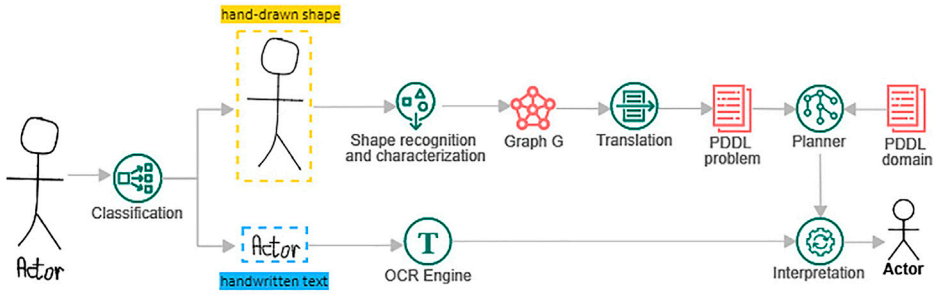

FIGURE 2
Overview of the recognition process.

requiring different techniques. Therefore, our process splits into two sub-processes to recognize text and geometrical shapes, respectively. Text recognition is performed *via* classical components-off-the-shelf OCR algorithms, while geometrical shape recognition is handled by our goal recognition algorithm described in [15]. Our approach then reconciles the outputs of both sub-processes once they terminate to reconstruct the global model element annotated with textual properties.

3.1.1 Classification

We developed a data efficient neural network classifier to distinguish the different elements composing the user's drawing into two categories: text and geometrical shapes. To address the lack of data, we opted for a feature-based neural network instead of an image-based NN since the former are lightweight and easy to train. First, we analyzed several inputs to identify features of detected geometrical shapes and text to be used in classifying a user's drawing. We then proposed a set of features to be computed from the data acquired using a training interface (see Section 3.3.1), and to use them in the classification. The list of the features is detailed in Section 3.3.

We then provided the NN classifier with a training data set of the appropriate network behavior in the form of labeled data. The training dataset consisted of a set of features (the extracted features from each category of the user's drawing) and a target (the corresponding category, i.e., text or geometrical shape). We used supervised learning where the network, provided with inputs, adjusted its weights and biases to move its outputs as close as possible to the targets.

3.1.2 Shape recognition and characterization

We use goal recognition to identify final geometrical shapes. This involves two steps. The first step consists in recognizing and characterizing primitive shapes with a simple shape detection algorithm. As described in [15], the approach only focuses on two primitive shapes: ellipses and straight lines. A polyline consists of a series of connected straight line segments. When a polyline is recognized, it is not characterized as a whole, but instead, each segment composing it is characterized individually and independently of the others. Circles are also recognized as a specific case of ellipses where the two foci are on the same spot (the center).

The rationale behind the algorithm's minimalist recognition strategy is twofold: first, this strategy reduces the time required to perform this step, hence, speeding up the complete recognition process. Second, most of the graphical elements employed in standard modeling languages and used by systems engineers can be simply expressed in terms of the two primitive shapes. This strategy considerably reduces the complexity of our goal recognition algorithm as its does not have to deal with multiple alternative ways of drawing the same model element. It allows the algorithm to deal with different drawing habits the same way. For example, a user can draw a rectangle in a single-line drawing (without lifting the pen from the surface of the screen) to represent the outer frame of a UML class, while a second user can draw four straight lines connected to their end-points.

Mathematically speaking, we consider the set S = (*ellipse*, *straight_line*) as the set of primitive shapes recognized by the algorithm. S is a minimal functionally complete set (by analogy with mathematical logic) as all other geometrical shapes can be expressed in terms of the two constituents of S.

Once primitive shapes are recognized, specific characteristics are extracted from them. Ellipses are tagged as being circles or not. A straight line is characterized by its four possible orientations O such that O = (*horizontal*, *vertical*, *diagonal_left*, *diagonal_right*). To compute straight line orientations, the raw angle between the two end points of a line is 'smoothed' to its closest remarkable $\frac{i\pi}{4}$-angle. Finer-grain fractions can be chosen for smoothing angles, but they would be less tolerant to drawing imperfections. For example, to sketch an operational actor, left and right legs could be characterized as 45° or 60° straight lines depending on the user's talent for drawing.

After the recognition process, we compute the position of every primitive shape relatively to their connection the other shapes composing the same element. We distinguish if a connection intersects a shape in the middle or at its end-

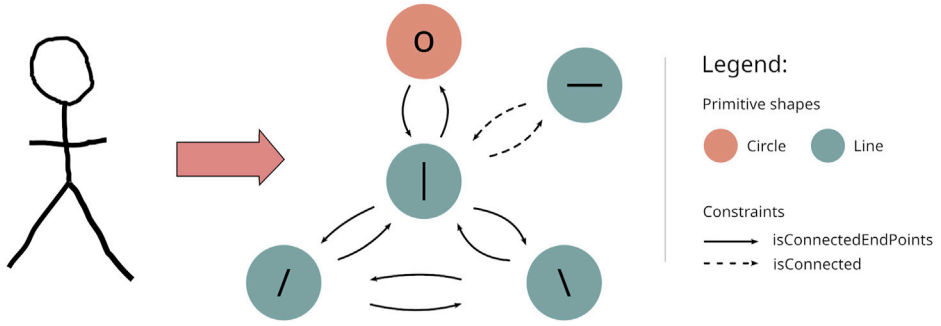

FIGURE 3
An actor being drawn (left side) and the resulting graph G (right side). Each edge is a relation between two primitive shapes connected at their end points or intersecting. The orange node represents the head of the actor. Green nodes are straight lines whose orientations are indicated by the node labels.

points. The set of possible intersections O is such that $O = (isConnectedEndPoints, isConnected)$.

The relation *isConnected* is bijective. It occurs when two primitive shapes are intersecting at the center:

$$\forall s_1, s_2 \mid s_2 \in isConnected(s_1) \Leftrightarrow s_1 \in isConnected(s_2) \quad (1)$$

The relation *isConnectedEndPoint* implies that the two elements are connected:

$$\forall s_1, s_2 \mid s_2 \in isConnectedEndPoint(s_1) \Rightarrow s_1 \in isConnected(s_2) \quad (2)$$

and is bijective as well:

$$\forall s_1, s_2 \mid s_2 \in isConnectedEndPoint(s_1) \Leftrightarrow s_1 \in isConnectedEndPoint(s_2) \quad (3)$$

The output of the first step is a directed graph $G = (V, E, l_v, l_e)$ where the set of vertices V corresponds to the set of primitive shapes composing a sketch, and the set of edges E corresponds to the connecting constraint between the vertices (see Figure 3). We apply two labeling functions. The vertex labeling function, l_v: $V \rightarrow \mathbb{L}^2$, decorates each vertex with a label denoting the nature (ellipse or straight line) and the distinctive feature (orientation for straight lines, nature for circles or not for ellipses) of the primitive shape corresponding to the vertex. The edge labeling function, l_e: $E \rightarrow \mathbb{L}$, decorates each edge with the corresponding connecting relation (*isConnected* or *isConnectedEndPoints*) that binds each pair of primitive shapes.

Listing 1: Predicates of the PDDL domain definition.

```
(define (domain BOARD)
(:requirements :strips :typing :equality)
(:types block shape)
(:predicates (isConnected ?x ?y - block)
    (isConnectedEndPoints ?x ?y - block)
    (onboard ?x - block)
    (removed ?x - block)
    (hasShape ?x - block ?z - shape)
    (hasOrientation ?x - block ?z - orientation) )
```

3.1.3 Translation into PDDL

The second step of the shape recognition sub-process consists in translating the graph obtained during the previous step into the Planning Domain Definition Language (PDDL) format [38]. PDDL is an attempt to formalise a standard to describe AI planning problems that is shared by various components-off-the-shelf AI planners [28, 39]. We use PDDL to formalize our drawing problem, describe the initial sketch, and describe the list of all the model elements deemed possible. This list constitutes the goal library, i.e., the set of the possible goals that our framework will consider when doing goal recognition. The PDDL domain definition contains the formalization of the drawing problem. Listings 1 and 2 are excerpts of the domain definition. It describes predicates that accept variables of two different types: blocks (made of polylines) and shapes (ellipses or straight lines). The coding of the predicates follows the relations given earlier.

Listing 2: A PDDL example of connecting a block to another one. Here, effects make use of conditions encoded with "when": Conditions are like preconditions, but if they do not hold in a state, the actions are still executed, but the conditional effect that does not hold will simply not be applied in the state.

```
(:action connectEndPoint
   :parameters (?x ?y - block)
   :precondition (and
      (not (= ?x ?y))
      (onboard ?x)
      (onboard ?y) )
   :effect (and
      (isConnectedEndPoints ?x ?y)
      (isConnectedEndPoints ?y ?x)
      (when (isConnected ?x ?y)
        (and
          (not (isConnected ?x ?y))
          (not (isConnected ?y ?x)) ))
    ) )
```

The domain definition also contains actions that can be performed by a user, or, more symbolically, by a drawing agent, to complete a sketch. We define *connect* actions to

complete an existing sketch with new shapes. For example, Listing 2 shows the definition of the *connectEndPoint* action to connect two polylines. We also define actions in our plan to support quick fixes [15]. A *remove* action consists in removing from a graph G a node and the edges that connect that node to other nodes of the graph. A *remove* action is interpreted as the primitive shape (represented by that node) has been incorrectly drawn and should not be considered as part of the sketch. *Update* actions consist in modifying a node or an edge of a graph G. An example of a node update action is changing an eclipse into a circle, or changing a straight line's orientation (*change-orientation*).

Every action also implements collateral effects on the shapes composing a sketch, according to the different relations formalized in Section 3.1.2. For instance, if the block A is connected to block B, then B will also be connected to A (Listing 2). We decided to encode collateral effects in the effects of the actions rather than using *axioms*[2] [40] as it was the case in a previous work [15]. The reason lies in the computing overhead that axioms yield, and in the scarcely availability of automated planners implementing them. Thus, the PDDL encoding of the sketching problem presented here is an evolution of a former implementation that was modeling the relative position between blocks in order to express their position on the board (e.g. `left-of, right-of` etc.). We decided to model the connections only between elements, as they are sufficient, along with their shape, to define the model elements used in the sketches. This simplification was dictated by the performances of the previous version of the model. The relative position between blocks needed to be adjusted for all the drawn blocks (this was done by using axioms) after any action. The current model is more compact, and produces shorter plans for certain models, also because of choice of having the property of bijectivity for the connections, as indicated in Eqs 1, 3. Of course the degrees of liberty implied by this modeling yield some ambiguity in the representations, but 1) some ambiguity would be present for any modeling choice, 2) in the context of software diagrams (e.g., UML), these modeling choices allow to represent all the sketches without ambiguity.

The formalization of the drawing problem and the goal library are generic and reused across different executions of the recognition process. Only the formalization of the initial sketch in PDDL is specific and is automatically generated from the graph obtained in the previous step. Listing 3 shows an example of a PDDL problem carrying the information of the initial sketch and a possible goal to reach. The initial state (lines 7–25) is generated automatically on the basis of the sketch currently drawn. The goal describes the positioning constraints required to build an operational actor.

Listing 3: An example of a PDDL problem generated from our approach. The mapping between the variables of the initial state and the variables of the goal is automatically carried out by the translation process.

```
(define (problem BOARD-actor)
(:domain BOARD)
  (:objects
      c1 v1 h1 dr1 dl1 h2 v3 h3 v4 h4 v2 - block
      line circle - shape
      none vertical horizontal diagonal_left diagonal_right - orientation)
(:INIT
  (onboard v1) (onboard h2) (onboard v3) (onboard h3)
  (onboard v4) (onboard h4) (onboard v2) (onboard h1)
  (isConnectedEndPoints v1 h2) (isConnectedEndPoints h2 v1)
  (isConnectedEndPoints v1 h3) (isConnectedEndPoints h3 v1)
  (isConnectedEndPoints h2 v3) (isConnectedEndPoints v3 h2)
  (isConnectedEndPoints v3 h3) (isConnectedEndPoints h3 v3)
  (isConnectedEndPoints v4 h4) (isConnectedEndPoints h4 v4)
  (isConnectedEndPoints v4 h1) (isConnectedEndPoints h1 v4)
  (isConnectedEndPoints h4 v2) (isConnectedEndPoints v2 h4)
  (isConnectedEndPoints v2 h1) (isConnectedEndPoints h1 v2)
  (hasShape v1 line) (hasOrientation v1 vertical)
  (hasShape h2 line) (hasOrientation h2 horizontal)
  (hasShape v3 line) (hasOrientation v3 vertical)
  (hasShape h3 line) (hasOrientation h3 horizontal)
  (hasShape v4 line) (hasOrientation v4 vertical)
  (hasShape h4 line) (hasOrientation h4 horizontal)
  (hasShape v2 line) (hasOrientation v2 vertical)
  (hasShape h1 line) (hasOrientation h1 horizontal) )
(:goal  (AND
    (onboard c1) (onboard v1) (onboard h1) (onboard dr1) (onboard dl1)
    (hasShape c1 circle) (hasShape v1 line) (hasOrientation v1 vertical)
    (hasShape h1 line) (hasOrientation h1 horizontal) (hasShape dr1 line)
    (hasOrientation dr1 diagonal_right) (hasOrientation dl1 diagonal_left)
    (isConnected v1 h1) (isConnected h1 v1) (hasShape dl1 line)
    (isConnectedEndPoints c1 v1) (isConnectedEndPoints v1 c1)
    (isConnectedEndPoints v1 dr1) (isConnectedEndPoints dr1 v1)
    (isConnectedEndPoints v1 dl1) (isConnectedEndPoints dl1 v1)
    (isConnectedEndPoints dl1 dr1) (isConnectedEndPoints dr1 dl1)
    (not (onboard h2)) (not (onboard v3)) (not (onboard h3))
    (not (onboard v4)) (not (onboard h4)) (not (onboard v2))) ) )
```

Based on the three inputs, we run the planner for the sketch being drawn by the user. To do it, we used the Fast Downward planning system [41], running the version of LAMA Planner [17] that participated in IPC 2011, and that has been integrated into Fast Downward's code. The anytime algorithm used in LAMA, to the contrary of the previous algorithms discussed above, does not continue the weighted A* search once it finds a solution. Instead, it start a new weighted A*-based search from the initial state. The planner then outputs several plans, with increasing quality (measured in the number of actions in the plan). Each plan is an ordered list of possible matches between the sketch being drawn by the user and the goals denoting the different model elements that could be recognized. The set of possible matches is ordered based on the degree of confidence of the match regarding the element currently drawn. The degree depends on the *distance* (in the plan) between the current sketch and the possible goal, i.e., the number of steps that would remain to finish drawing the element completely.

3.2 Implementation

Figure 4 pictures the BOARD-AI main interface. The interface is developed using Web technologies (HTML, CSS, and JavaScript) so it can be used remotely and it can be run on any interactive pen display devices, ranging from tablets to large screens equipped with stylus. We adopted a minimalist style where the entirety of the screen can be used to draw a model so that users can fully focus on the sketching activity without being disturbed by an overloaded interface. The main area is an HTML5 Canvas for drawing model elements. A toolbar provides some useful features, such as undo/redo, page management, different edition modes (drawing, erasing, and

2 Axioms are specific actions applicable to a state, but they do not contribute to the evaluation of the distance between the current state and a goal.

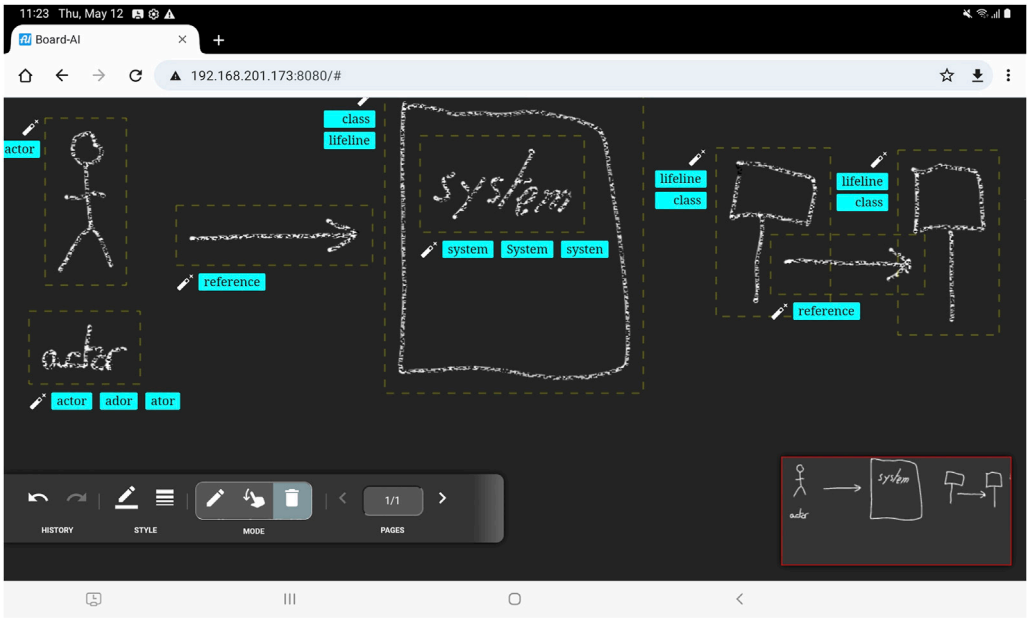

FIGURE 4
Screenshot of the BOARD-AI interface running on a Samsung Galaxy S6 Lite. Its minimalist interface features a wide area to sketch model elements, a collapsible toolbar, and an outline. Blue bubbles along with drawn sketches provide hints about the model elements to recognize.

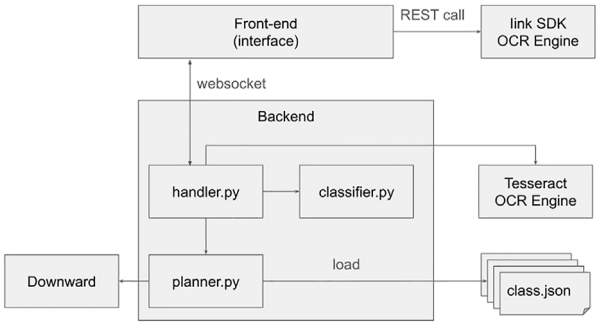

FIGURE 5
Overview of the BOARD-AI architecture.

selection), and two options to customize the thickness and the color of the drawing. The toolbar can be collapsed to maximize the drawing area. Besides these features, a chalk effect is applied to replicate the writing on a blackboard.

Two distinct interaction modalities are used to interact with the screen. Drawing is only possible using the stylus while touch gestures enable the user to navigate across the interface. Zooming in and out is achieved by pinching the screen. Single finger panning can also be used to navigate within the Canvas when it is scaled up. When the interface is scaled up, an outline at the bottom right of the screen shows the visible area.

When the user starts drawing, the recognition process is performed. Under the hood, the recognition engine consecutively identifies and characterizes the primitive shapes drawn by the user, invokes the classifier, and performs text or shape recognition according to the output of the classifier. Visual hints are given in the shape of bubbles accompanying the elements to recognize. Clicking on a visual hint results in converting the partial drawing into the suggested model element or text. If the partial drawing is updated by the user, the suggestions are updated as well.

Multiple shapes and/or textual annotations can be recognized at the same time. Hints to geometrical shapes are provided by the Fast Downward planning system and are updated according to the anytime algorithm used. A limit of the three best suggestions is kept and displayed along with the partial drawing. Recognizing textual annotation is done using third-party OCR engines. Our implementation currently supports both MyScript Interactive Ink SDK (iink SDK) and Tesseract OCR engines. When using iink SDK, multiple suggestions are provided and displayed to the user. A metric distance is then applied to reconcile model elements with textual annotation. As an example illustrated in Figure 4, textual annotations will be respectively converted into class or actor names.

Figure 5 describes the architecture of BOARD-AI. It follows a client-server architecture where the back-end (in Python) is responsible for calling the different services for shape recognition. It consists of three main modules. The *handler* module makes the link with the interface. It starts a Web-socket server to communicate with the front-end. Shape recognition being incremental, the web-socket communication is used to update the interface every-time a more optimized plan is found. A *Classifier* module built on top of *numpy* distinguishes geometrical shapes from textual annotations. The *planner* module is responsible for translating sketched elements into PDDL as discussed in Section 3.1.3. Models elements to be recognized are structured as JSON objects (see Listing. 4 as an example) and stored in separate files. When the *planner* module starts, it first loads the different files and then translates each JSON object into PDDL problem templates as shown in Listing 3[3].

Listing 4: An example of the definition of the goal in JSON for recognizing a UML class.

```
{
  "name": "class",
  "shapes": [
    {
      "name": "h1",
      "type": "line",
      "direction": "horizontal"
    },
    {
      "name": "h2",
      "type": "line",
      "direction": "horizontal"
    },
    {
      "name": "h3",
      "type": "line",
      "direction": "horizontal"
    },
    {
      "name": "v1",
      "type": "line",
      "direction": "vertical"
    },
    {
      "name": "v2",
      "type": "line",
      "direction": "vertical"
    }
  ],
  "areConnectedEndPoints": [
    {"s1": "h1", "s2": "v1"},
    {"s1": "h1", "s2": "v2"},
    {"s1": "v1", "s2": "h2"},
    {"s1": "v2", "s2": "h2"}
  ],
  "areConnected": [
    {"s1": "h3", "s2": "v1"},
    {"s1": "h3", "s2": "v2"}
  ]
}
```

Upon requesting a recognition of a sketched elements, the following actions are executed. First, the *classifier* module starts by classifying the sketched element as text or shape elements. If a sketched element is classified as text, text recognition is performed by an OCR engine. Depending on the selected OCR engine used (being Tesseract or iink SDK), text recognition is performed by the back-end or the front-end. However, if the sketched element is classified as a geometrical shape, shape recognition is performed by the *planner* module. This module translates the sketched elements into PDDL and invokes multiple instances of Fast Downward as parallel sub-processes. As soon as a plan is generated by one sub-process, it is broadcast to the front-end interface through the *handler* module.

Running multiple instances of Fast Downward in parallel speeds up the recognition process as suggestions of shapes are provided in the interface as soon as they arrive. In our tests, recognizing multiple model elements (mixing geometrical shapes and text annotations) as the one pictured in Figure 4 takes between 500 milliseconds and 1 s. However, it is computationally heavy and requires a proper server architecture to reduce the computational time.

3.3 Model training

This section describes the training of the NN classifier we used. We chose to set up a NN based on specific features rather than on exploiting images as the first solution is more cost-effective and requires less data than the second one. The section first details the training interface we developed, then it discusses the data acquisition and the features that were extracted to correctly train the classifier. Finally, it presents the validation of the training.

3.3.1 Training interface

Figure 6 is an overview of the training interface. It has been used to train the NN to learn how to differentiate geometrical shapes and text. The interface shares several similarities with the interface of BOARD-AI. It was developed using the same technologies (HTML, CSS, and JavaScript) and relies on the two same interaction modalities. We chose to separate the two interaction modalities so as not to lead to a bias during the training. Finger drawing may result in a bad training of the NN. Besides, styli seem to be the most natural way of drawing onto a screen and provide a nicer user experience for text annotation than finger drawing. During the experiment, we observed that the participants naturally use one hand to hold the stylus while using the second one to navigate within the HTML5 Canvas.

The training interface contains three areas. The main area is an HTML5 Canvas for drawing model elements, as it is done in the interface of BOARD-AI. Both sidebars contain indications for the user to understand how to use the training interface. The interface has been used as an experimental platform to collect users' drawing data so as to train neural networks to predict which parts of a drawing relate to text and which parts relate to

3 The templates are later filled with the missing part describing the initial state of the problem.

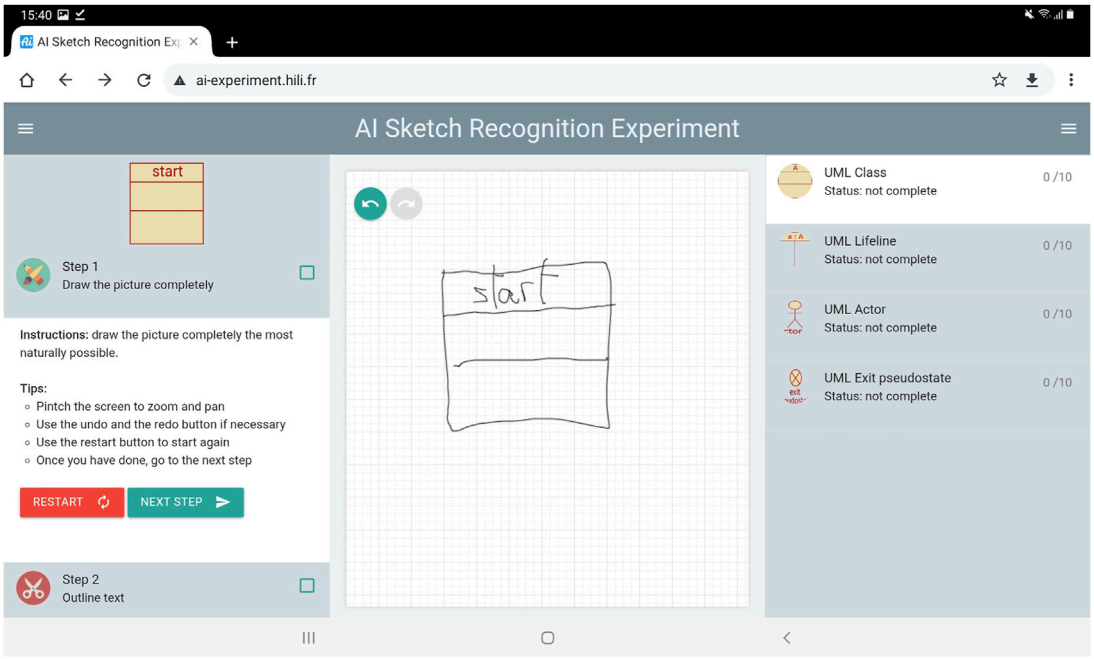

FIGURE 6
Overview of the training interface. It is used to train the neural network. The left sidenav shows indications to guide participants through the training. The right sidenav shows the progress of the participant. The main central area accepts pointer events to draw and touch events to navigate within the HTML5 Canvas.

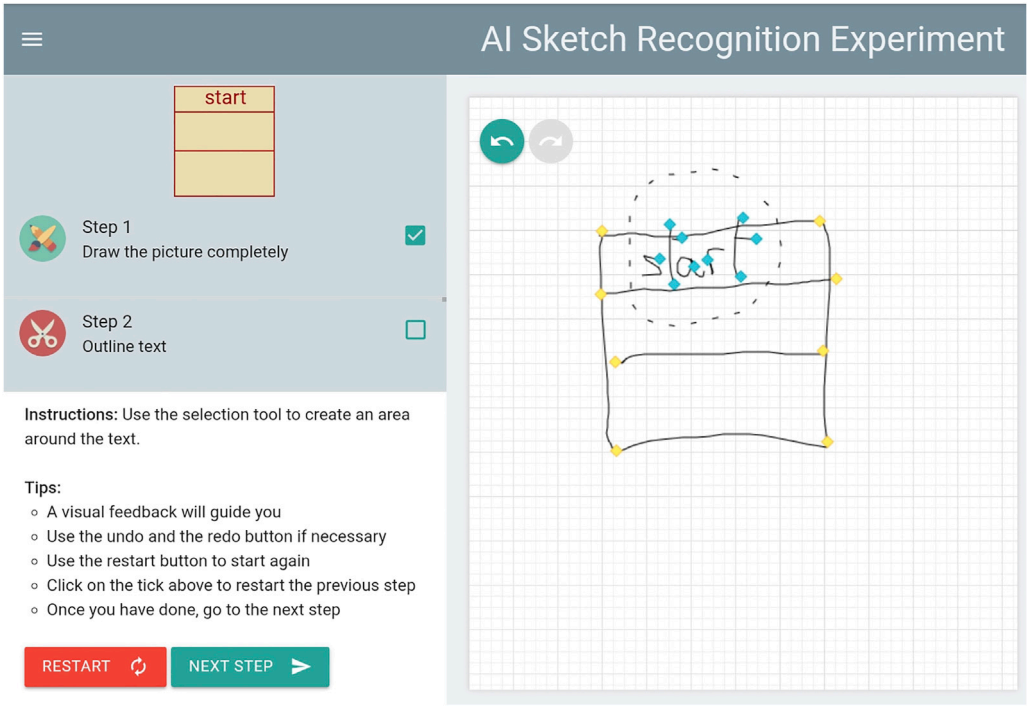

FIGURE 7
Upon completing a sketch, participants are asked to select (using a Lasso Tool) the text only. Sharp corners are shown in yellow. They turn blue once they have been selected.

geometrical shapes. Using the interface, users are invited to perform two successive steps: *drawing* and *text selection*.

Drawing

During the first step, users are invited to draw graphical model elements as they appear on the top left corner of the interface. Each drawing is stored as a set of paths. A path starts when the user touches the screen with the pen and ends when the pen is lifted from the surface of the screen. A path is a collection of points acquired by the device during this timeframe. A point corresponds to a pixel drawn on the screen. It contains a xy-coordinate on the screen. The frequency rate of events fired to detect that a point is drawn on the screen is device-specific and depends on the hardware implementation of the tablet [42]. In addition to its xy-coordinate on the screen, a point also contains various metadata collected from the stylus. Specifically, we collected the pressure applied on the stylus, the plane angles *tiltX* and *tiltY* between the stylus and the surface, and the timestamp when the pixel is drawn on the screen. We decided to keep several pieces of metadata to find which ones are relevant to efficiently train the neural network, but also for being able to precisely and accurately replicate the experiments offline, and to understand the user's habits in terms of drawings. This understanding will lead to new experiments in the future.

Text selection

Once the user completes his/her drawing, a line recognition algorithm detects sharp corners. Sharp corners are particular points of a path denoting marked directional changes. This step is used by our algorithm to transform a path into multiple straight lines. During the second step, users are invited to select sharp corners belonging to any textual part of the drawing using a *Lasso* tool (see Figure 7). This second step is important for the classification process to distinguish text from geometrical shapes.

3.3.2 Data acquisition

We used the interface to collect data to train the NN. Each participant was asked to draw ten times, four graphical elements composed of one of four chosen elements coming from UML (class, actor, lifeline, and pseudo-state) and a randomly chosen English word. In practice, this represents, for each participant, a total of eighty training samples for the network (forty for each class, text or shape).

We took different measures so as not to introduce any bias in the experiment. All experiments were performed on the same device, a Samsung Galaxy S6 Lite tablet equipped with an active pen stylus. Finger drawing was disabled. Random words were chosen from a dataset of the most commonly used English words to add variability in the training process. We developed a flip mode for left-handed users where both sidebars are flipped. This mode has been developed so that the drawing directives always appear on the opposite side of the hand holding the stylus and are not covered by the hand. As a final measure, the four types of model elements were presented to the participants in a random order to prevent "muscle memory".

3.3.3 Feature design

The data acquired using the training interface described above was pre-processed and analysed in order to keep only the relevant pieces of information. To use our NN classifier, we identified the features to be passed as inputs to the network. Features were chosen to be independent of the writing speed and also scale independent, i.e., independent of the size of the user's drawing. For these reasons, data such as the number of points (depending both on the acquisition capabilities of the tablet and on the speed of the user's writing) were removed. Other data such as the length and the width of the bounding box was not used directly but computed to form new measures relevant to the network. After conducting the analysis on the data, the chosen features are:

- the number of sharp corners. We can imagine that this number is greater for text.
- the bounding box ratio: It is computed as the ratio between the width and height of the bounding box. The bounding box is the smallest rectangle encompassing all points. We can imagine that text would tend to have horizontal boxes and shapes more vertical boxes.
- the longest segment ratio: It is computed as the longest corner segment divided by the longest side of the bounding box. A segment is defined as a line between two consecutive points, and a corner segment is defined as a segment between two consecutive corners. We can imagine that we are more prone to find longer segments in geometrical shapes e.g. boxes, arrows, actors rather than in text.
- the total segment ratio: It is computed as the sum of all corner segments divided by the longest side of the bounding box. We can expect that text will tend to have a greater ratio.
- the minimum angle corner: It corresponds to the minimum angle computed among all angles found between two corner segments. We can expect that shape will tend to have a greater minimum angle.

3.3.4 Network design

The starting proposed structure for the NN classifier corresponds to a multilayer fully connected network, and it is composed of:

- one input layer of size five since we have five features.
- one output layer that represents one of the categories; it must be one or zero for each geometrical shape or text, respectively.
- one to three hidden layers between the input and the output layers. Besides these three layers, the size of each layer varies from five to ten neurons.

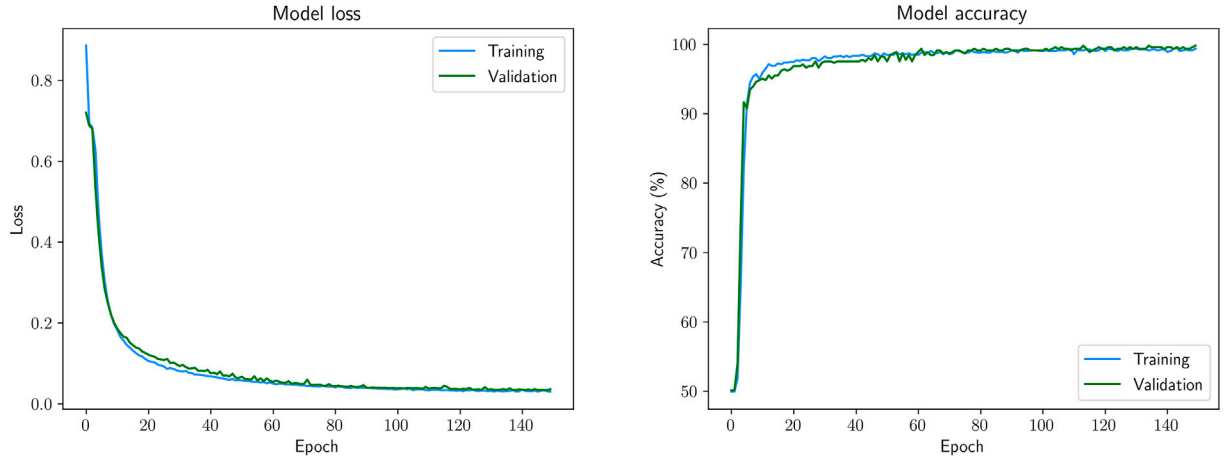

FIGURE 8
Model loss and accuracy on training and validation data for the final structure network.

On the one side, the rectified linear unit function [43, 44] was chosen as an activation function for the hidden layers. On the other side, to guarantee that our network output is between 0 and 1, we used a Sigmoid activation function for the output layer.

The final structure of the network was chosen after the training and validation step. It is described in the following section.

3.3.5 Training and validation

Our dataset is composed of 1,342 entries (after removing 18 invalid entries) coming from 17 different users who participated in the user experiment described earlier. The dataset was divided between 67% training and 33% validation.

Training a network consists in finding the best set of weights to map the elements to be classified to their respective categories. To evaluate a set of weights, we must specify a loss function. This function is used by an optimization algorithm to estimate the loss of the model, update weights and reduce the loss on the next evaluation.

To train our network, we used the Adam optimization algorithm and we use cross-entropy as the loss function for our binary classification problem. This process was run for 150 epochs (the number of iterations through the dataset) and a batch size of 10 (number of training data considered per epoch).

To tune the hyperparameters (number of hidden layers, number of neurons for each hidden layer) we conducted 500 experiments. For each experiment, the number of hidden layers was randomly chosen between one and three and for each hidden layer, the number of neurons was randomly chosen between five and ten. The final structure network showing the best results was composed of two hidden layers of sizes eight and five nodes respectively. Figure 8 present the performance of the network over time during training. On the one hand, we can observe that the model loss has comparable performance on both datasets. On the other hand, the model accuracy plateaus indicate that the model did not underfit and that the validation accuracy did not diverge from the training accuracy, indicating that the model did not overfit. The validation stage showed that the final structured network has a good performance and a 99.77% prediction accuracy.

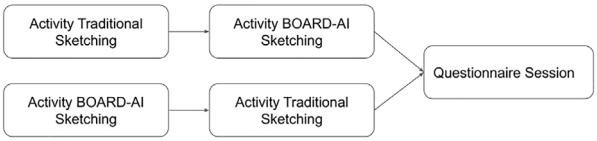

FIGURE 9
Exploratory Study Design and data gathering techniques.

In future work, we would like to consider if training on a subset of the available shapes of our training data would still allow for good classification performance even on unseen shapes.

4 Evaluation

4.1 Aims and research questions

BOARD-AI was designed to facilitate sketching system engineering models. For its implementation, BOARD-AI was trained using data collected from human draws and annotations (see Section 3.3). We present now the results of evaluating

TABLE 1 Data gathering instruments.

Data gathering technique	Description	References
Questionnaire	Questionnaire with 22 questions. This questionnaire is composed of 3 preliminary questions about the user's background, the Usability Scale (SUS) [45] translated to French, i.e. 10 closed questions, plus 7 closed and 2 open questions inquiring the user about the functionalities of the tool	https://osf.io/v45c8/
Participants Consent Form	Consent form to be signed before participating in the experimental study	https://osf.io/v45c8/
Protocol Document Activity Group_1	Document facilitated to the Group_1, which started sketching using a traditional method, and then using BOARD-AI	https://osf.io/qutx3/
Protocol Document Activity Group_2	Document facilitated to the Group_1, which started sketching using BOARD-AI, and then a traditional method	https://osf.io/g36rz/

BOARD-AI with final users. This evaluation permits to understand how BOARD-AI supports engineers in sketching system models, and allows us to identify future improvements of the tool. Specifically, four research questions drove the evaluation:

RQ 1. Does BOARD-AI facilitate the modeling process compared with traditional methods?
RQ 2. How helpful are the tips and shape suggestions provided to the users, so as to support sketching system models?
RQ 3. Would a false positive in recognizing a geometrical symbol or a textual label affect the user the same way?
RQ 4. How usable is the BOARD-AI tool?

4.2 Methodology and data collection

To address the research questions proposed for the evaluation, we conducted an exploratory study. This is the recommended methodology for studying a phenomenon when there is insufficient prior research to establish concrete hypotheses. In this case, we wanted to study the effects of using BOARD-AI in supporting engineers on sketching their system engineering models compared with other existing more traditional methods. For this purpose, we prepared two different protocols for two groups of users (Group_1 and Group_2). Both protocols consisted in a sequence of two activities that participants had to follow:

- Activity Traditional Sketching: Users are asked to sketch a system model using paper and pencil or whatever tool of their choice they usually use for sketching models.
- Activity BOARD-AI Sketching: Users are asked to sketch the same system model using BOARD-AI.

The first group of participants (Group_1) started with the Activity Traditional Sketching and continued with the Activity BOARD-AI Sketching, whereas the second group (Group_2) of participants performed these activities in the reverse order. Combining the use of traditional sketching methods and BOARD-AI allowed the users to be able to compare the two methods for conducting similar activities and, while having two different groups isolated the effect of the novelty of using BOARD-AI in such an activity. Figure 9 shows a schema of the experimental procedure conducted.

Different data gathering techniques were used to collect data about the two activities at different moments. First, the participants were asked to fill in a consent form for participating in the experiment. Then, a researcher facilitated them with a document explaining the activities to be conducted and how to access the BOARD-AI tool. Then, the participants completed the two activities with no time limitations. Finally, the participants answered a final questionnaire containing 22 questions.

The first three questions are dedicated to determine the profile of the participants and their experience in the modeling domain. The next 10 questions (from question one to question 10) correspond to the SUS standardized questionnaire [45], designed to measure the usability of the tool. Then, questions 11 and 12 are dedicated to compare BOARD-AI with other traditional methods; questions 13 to 17 refer to aspects related with the tips and graphical help offered by BOARD-AI, and questions 19 and 20 (the latter is a supplementary question for non expert users only) ask about which aspects of the tool are considered the best, and what are those that should be improved. Except questions 19 and 20, which are open questions, the rest follow a Likert scale from 1 to 5, were 1 is "completely disagree", and five is "completely agree". Table 1 shows the different data gathering techniques that were employed as well as the links to the instruments used.

4.3 Participants and analytical methods

As a sampling method, we decided to follow convenience sampling for selecting participants. In this case, 12 participants from the University of Grenoble Alpes, engineers, experts and

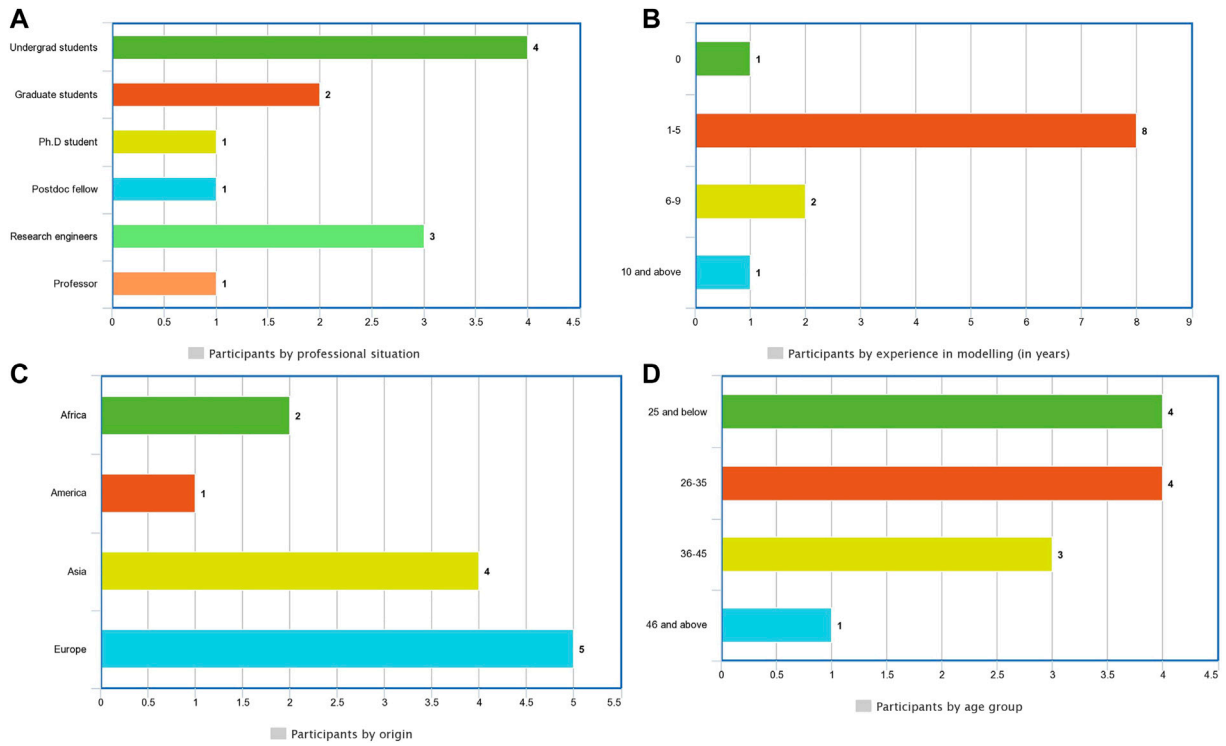

FIGURE 10
Overview of the participants in terms of professional situation **(A)**, experience in modeling **(B)**, origins **(C)**, and age groups **(D)**.

non experts in software and/or system modeling design, were invited *via* e-mail and all accepted the invitation. Six participants were randomly assigned to Group_1 and the other six to Group_2. All participants received an e-mail indicating the place and the time for participating in the study. Participation was voluntary and no reward was proposed.

Coverage/representativeness of the user base being an important concern in any user experiment, the 12 participants have been invited to have a sample representative of the population interested in using sketch-based tools. All participants but one had previous practical experience in modeling or had been taught modeling during their academic courses. Although all participants shared the same affiliation at the time of the user experiment, their profiles presented several variations, in terms of experience in modeling, CASE tools/sketch tools they are comfortable with, employment situations, age groups, and origins. Figure 10 details the different participants.

For answering the RQ1 about how BOARD-AI facilitates the modeling process compared with traditional methods we analyzed the answers to questions 11 and 12 in the Questionnaire. For answering RQ2 about how helpful the tips and shape suggestions of BOARD-AI are, we analyzed the answers to questions 13 to 16 in the Questionnaire (both included). For answering RQ3 about whether a faulty text recognition is less important than an incorrect shape recognition, we analyzed the answers to question 17 specifically. In all cases, we calculated the percentage of answers given by the participants of the experiment between 1 and 2 (badly evaluated), 3 (neutral) and 4–5 (well evaluated). Questions 1 to 10 (corresponding to the SUS questionnaire) were analyzed following the instructions provided in its design for answering RQ4 regarding the usability of the tool. Finally, we qualitatively analyzed the answers given by the participants to improve the tool, and classified them according to the different emergent topics. These answers were used to complement the data collected through the questionnaire.

4.4 Results

The participants were asked to perform similar tasks using two sketching methods. Comparing BOARD-AI to more traditional methods and to engineers habits permits to assess the usability of the tool, its performance, and eventually the trust that users are willing to put in an AI-based tool.

First, regarding RQ1, the answers to the questions 11 and 12 underlined the simplicity of using BOARD-AI. 83.4% of the

participants (scores 3–5) considered that sketching with BOARD-AI was easier (or of the same difficulty) than traditional methods. This result is supported by the answers to question 12 about the confidence they had in the sketch they made: 75% of the participants considered that using BOARD-AI the resulting sketch was less or equally prone to contain mistakes or errors than using the other method—66,7% answered that they trusted the BOARD-AI sketch more in avoiding potential errors.

Second, and regarding RQ2, participants' answers to the questionnaire suggest that the support provided by BOARD-AI through tips and shape suggestions are positive and valued by the end-users. On the one hand, results from analyzing questions 13 and 14 indicate that no participant said that the tips, the shape suggestions, and the toolbar provided by the modeling interface were improper (that would have been scores 1–2). 75% of the participants gave high scores [4, 5] to the suitability of the suggested outline to the sketch being drawn, while 41.7% approved with similar scores the proposed toolbar. On the other hand, the analysis shows that users see in the suggestions an element of trust in the tool that facilitated their modeling process and their confidence on the result. This is supported by data collected from questions 12 and 15, which shows that all but two participants trusted the suggestions, that were judged appropriate. Even more, 2/3 of the participants indicated that the completion suggestions were appropriate to their intentions, easing and quickening the sketching job. These results are also supported by the qualitative data collected, which indicates that participants appreciated the shape and text recognition as a mechanism that could facilitate collaborative work: "Text recognition is new feature - the tool will help team to be collaborative" (Participant 11). Also, the analysis shows that suggestions on graphic elements helped them to create their models: 83.4% of the participants said that including the AI-suggested graphic elements in their final sketch helped them greatly to complete the task (scores 4–5) (Question 16).

Third, and regarding RQ3, participants were asked to answer to a specific question to see whether faulty text recognition had less impact than faulty shape recognition. However, there is not a preferred mechanism for suggesting among the participants. 41.6% of the participants expressed a clear preference for trusting a system that has a good accuracy in recognizing drawn shapes, while 41.6% prefer trusting systems that perform a good text recognition (Question 17). This indicates that a widely accepted AI-based tool should progress on both aspects.

Questions 1 to 10 were used to determine the SUS score associated with the BOARD-AI user interface. The results of the SUS score in our study was 65.2%. This indicates an *OK* rating for BOARD-AI, as it is currently designed. However, this score varies depending on the experience of the participants. We noted that the mean of the SUS for less experienced participants (students) is 61.7%, and for more experienced participants (engineers, MSc, post-docs) it raises at a value of 68.75%. Thus, experienced system engineers gave an evaluation of the BOARD-AI interface and usability more towards a *Good* rating than less experienced participants. This can depend on the easiness to adapt to a new modeling interface for experienced users, who used other tools in the past, on the contrary of students, that are still learning to master more traditional tools. This result is complemented by the qualitative data collected. Participants found BOARD-AI easy to understand and to use. Participants especially value its simplicity, i.e. "It is simple to use" (Participant 4), or "Easy to separate colors of different concepts" (Participant 4).

However, and despite of the positive aspects of the tool, participants identified three main aspects to be improved. First, they highlighted that the tool is slow responding to the text and shape recognition, which could be improved if the tool is uploaded to a high performance server, and by other implementation adjustments. Second, participants indicated that the text recognition engine needs some improvement. They found that not all the texts were correctly identified. Since BOARD-AI relies on a third-party software for the text recognition, this is something that could not be controlled in this first evaluation. However, future work will analyze other possible solutions that could improve the text recognition process. And third, the participants identified that the deletion and help options were not enough intuitive and should be improved. This will be considered in future development of the UI.

4.5 Threats to validity

We identified several threats to validity we list below. First, the low number of participants and their shared affiliation to the same institute at the time of the user experiment can affect the validity of the results. Nevertheless, as mentioned above, we believe that the participants represent a sample representative of the population interested in using sketching tools and each participant differs from each other with respect to his/her personal background (experience in modeling, tools used during his/her past experiences, employment situation), origin, and age group.

Another limitation of the present study relates to the alphabet used. The NN has only been trained with models using the Roman alphabet, hence limiting the broad applicability of the approach. Regarding the user experiment, all participants were proficient in writing using the Roman alphabet and use it daily at their professional workplace, although half of the participants were native to other writing systems. However, it is worth noting that we did not observe any variation in the results of the user experiment based on this criterion, and that no participant stressed out this point when filling the questionnaire.

Finally, RQ3 relied on the hypothesis that a faulty recognition when recognizing text has less impact on the users' confidence

TABLE 2 Existing approaches for natural sketching.

	MyScript Diagram	OctoUML	Bresler et al. [10, 11]	FlexiSketch	Lank et al. [5]	Tahuti [7]
Platform	Web and Desktop	Web	.NET framework[a]	Android	Desktop	Desktop
Open source	✗	✓	✗	✗	✗	✗
Recognition						
Scope	Flowcharts, organizational charts, mindmaps	Class diagram only	Flowcharts, automata	Adaptable through type promotion	UML class, sequence, usecase	UML class
Algorithm	Proprietary	Geometrical shape detection[b]	classifier and segmenter	Geometrical shape detection[c]	classifier and segmenter	Geometrical shape detection
Sktech recognition	Basic geometrical shapes	Basic geometrical shapes	Flowcharts and automata symbols	Complex geometrical shapes	UML glyphs	Basic geometrical shapes
Bulk recognition	✓	✓[d]	✓	✗	✓	✓
Handwritten text recognition	✓	✗	✓	✗	✓[e]	✓[e]
Incremental recognition	✓	✓	✗	✓	✗	✓
Explainable results	✗	✗	✗	✗	✗	✗
Performance						
Accuracy	Relatively accurate	Relatively accurate	Accurate	Relatively accurate	Unknown	Unknown
Speed	Relatively slow	Moderately fast	Fast	Relatively fast	Unknown	Unknown

✓ = available ✗ = not available.
[a]Experiments have been implemented in C# but there is no mention of any implementation.
[b]Based on PaleoSketch [9].
[c]Based on a Levenshtein distance algorithm.
[d]With some restrictions.
[e]In both work descreibd in [5, 7], text is identified but not recognized. The use of third-party tools, e.g., OCR, engines is suggested.

than an incorrect interpretation of the modeling intent. The hypothesis is based on the fact that quick corrective actions can be taken in the event of incorrect text recognition (e.g., resorting to virtual keyboards to correct only the fragments of text incorrectly recognized). However, no strong agreement to this question came out, as half of the participants agreed or strongly agreed while half of the participant disagreed or strongly disagreed. One possible interpretation of this result was that, at the time of the user experiment, we did not provide participants with the aforementioned corrective actions to quickly fix faulty text recognition. Providing such corrective actions (with the help of virtual keyboards, physical keyboards, or speech recognition) could have tip the scale in favor to a good shape recognition accuracy over than a good text recognition one.

5 Related work

Different attempts have been done to design and implement robust online sketch recognition algorithms (see Table 2) [5] designed a online recognizer based on a segmentation algorithm for hand drawn UML diagrams sketched on electronic whiteboards. We followed the same principle of image acquisition where points are collected from the instrumented drawing surface and are converted into strokes and straight lines. However, timing information are used for the segmentation process while we chose not to use this data as it is highly dependent on the user and his/her drawing habits. Besides, the approach addresses the recognition of UML symbols and characters using the same segmentation technique. In our approach, we assume that both text and geometrical shapes belong to two different classes of problems and therefore require two different processings. Relying on goal recognition also preserves explainability as the user is not left clueless when the recognizer's outcomes do not correspond to the user's intent.

Bresler et al. [10, 11] propose a recognition method to recognizing flowcharts and finite automata. They use a segmentation and classification approach to separate text and symbols. We share the same rationale in our approach as text and shapes need separate techniques to be recognized. Text recognition relies on *Microsoft. Recognizers.Text*, a module of the *NET framework* ecosystem. Experiments conducted by the authors show that the recognition is fast and accurate. The approach can be generalized to any diagram consisting of symbols connected by arrows, but it requires large amount of data to train the classifier.

Tahuti [7] also addresses online recognition of UML diagrams. We share the same approach of expressing complex sketches in terms of geometrical properties and of primitive

shapes it contains. Like Tahuti, we strictly reduce the set of primitive shapes we use to a set sufficient to express any modeling elements for common modeling languages. Tahuti focuses on UML class diagrams. We tend to be more generic with our goal recognition approach where we can define library of goals to describe the model elements of various modeling languages. Tahuti supports handwritten text annotation with some limitations. Text is merely identified but not recognized. Its identification depends on positioning constraints that are specific to class diagrams. Text should be contained in the class or appear next to it. Our approach based on NN does not constrain the positioning of the text with regards to the model elements it annotates and tends to be more generic. Yet, once the classifier classifies a drawing as text, its positioning relative to the model element it annotates is expressed at the conceptual level and is part of the goal library.

SketchREAD [8] is a multi-domain sketch recognition engine using Bayesian networks to improve the recognition process. SketchREAD also reasons in terms of geometrical abstractions of a user's drawings and decompose complex sketches into strokes. It does not require any training data and only needs the description of the sketches in terms of subshapes and constraints between them. Therefore, it tends to be more generic than Tahuti, as it can be adapted to various domains. Our goal recognition approach shares the same philosophy, based on the definition of various libraries of goals, depending on the targeted modeling language. One major objective of using goal recognition is to preserve explainability of the outcomes of the modeling assistant to the user. Besides, SketchREAD does not seem to support handwritten text annotation as we do in the present study.

MyScript [20] is a leading company in the domain of handwriting recognition. It features *MyScript Diagram*, a natural sketching tool used to create various kinds of charts from flowcharts to mindmaps. Ten primitive shapes and connectors are recognized, and text recognition is supported in multiple languages. MyScript runs on desktops or in the cloud. The recognition algorithm remains proprietary and recognition can be done remotely (on a subscription basis) or on-device. Compared to the other solutions, MyScript Diagram does not need to rely on other interaction modalities (such as voice recognition or virtual keyboard) to recognize shapes and text in a simultaneous way.

OctoUML [12, 46] is the prototype of a modeling environment that captures UML models in a free-form modeling fashion and in a collaborative way. It can be used on various devices, including desktop computers and large interactive whiteboards. Sketches are then converted into a graphical UML notation. OctoUML supports class and sequence diagrams. It uses a *selective recognition* algorithm to support an incremental formalization.

OctoUML relies on PaleoSketch [9], a recognition algorithm capable of recognizing eight primitive shapes (lines, polylines, circles, ellipses, arcs, curves, spirals, and helixes) and more complex shapes as a combination of these primitive ones. By recognizing more primitive shapes than other low-level recognizers, PaleoSketch intends to recognize domain-specific shapes that could be indescribable using other methods. The drawback is that it consumes time to recognize more primitive shapes. In our tests, we observed that recognizing shapes takes on average 500 ms and up to 1 s, both of which are noticeable to the user. Besides this condition, the rationale behind recognizing more elementary shapes is elusive as some shapes (helixes, waves, spirals, *etc.*) are never used in modeling languages, specifically in systems engineering. In our approach, we took the opposite stance by choosing to recognize only a few primitive shapes (lines, circles, and ellipses) and to use plan recognition to identify model elements as any combination of these primitive shapes. The three primitive shapes are indeed sufficient in Model-Based System Engineering (MBSE) to recognize most modeling elements drawn in the most common modeling languages and to reduce the number of primitive shapes that need to be recognized to speed up the recognition process. In our tests, recognizing complex shapes (e.g., an operational actor made of four straight lines and one circle) fell under 100 ms, which is barely noticeable to the user.

FlexiSketch [19] is a diagram modeling tool available on Android platforms. In FlexiSketch, a user can sketch model elements and later promote them as types than can be easily re-used. Once a graphical sketch has been associated with a model element, similar sketches are automatically recognized. This allows for adapting FlexiSketch to new graph-based modeling languages. The FlexiSketch's recognizer relies on an adapted version of a Levenshtein string-distance algorithm. The recognition is relatively fast and accurate.

We note that among the different solutions, only MyScript Diagram and the recognition of Bresler et al. [10, 11], provide seamless handwritten text annotation recognition capabilities. In [5, 7], the use of OCR engines is suggested but not seamlessly integrated into the respective tools. OctoUML allows the users to add textual properties through a physical keyboard or *via* voice recognition which requires the users to first recognize and formalize the model elements before adding text. In FlexiSketch, textual properties of model elements are only set using the Android virtual keyboard. In our previous work [47], text annotation could also be attached to model elements after they have been recognized like in OctoUML, through a draggable virtual keyboard and voice recognition. Providing a support for mixed text and geometrical sketch recognition is an objective of the present study.

Finally, none of the aforementioned solutions provide explainable outputs. Some work such as FlexiSketch provide alternative suggestions, and MyScript Diagram can provide word suggestions during text recognition. But none of these solutions can explain why an element has been recognized in the first place. In our approach, the output of the recognizer is

completely explainable. The user is informed of which part of a modeling language is being recognized (the primitive shapes composing the modeling elements), and what remains to be drawn using visual feedback.

6 Conclusion

This study presents an approach for sketch recognition of systems engineering model elements combining the benefits of Machine Learning (ML) and Automated Planning. Compared to existing ones, this approach is able to recognize model elements annotated with text supports while preserving the explainability of the outcomes of the sketch recognizer. To achieve this result, the approach relies on ML and on a trained Neural Network (NN) to separate, upstream from the global recognition process, handwritten text annotations from geometrical shapes, as the two belong to two different classes of problems and require different recognition techniques. Component-off-the-shelf Optical Character Recognition (OCR) engines are indeed well suited for text recognition while plan and goal recognition techniques permit our system to recognize a sketched element even from a partial drawing.

In our previous work [14, 15], we detailed the adaptation of plan and goal recognition techniques for sketch recognition. In the present study: 1) we complemented the approach with ML, 2) we integrated two OCR engines (namely Tesseract and iink SDK) to seamlessly recognizing text annotations in model elements; 3) we improved our original implementation; and 4) we reformulated the planning domain to be lighter while adopting an *anytime* algorithm to produce faster plans with incremental quality.

It resulted in the definition and the implementation of a Human-machine interface named *board-ai*, which, compared to our initial prototype [15], now supports incremental recognition of multiple sketches in parallel mixing geometrical shapes and textual annotations. The validation stage used to classify the sketches gave us good results for the NN and a prediction accuracy of 99.77%.

We finally assessed the usability of the Human-machine interface for Systems Engineering modeling. Thus, results from the human data permitted an evaluation that helped us to understand how BOARD-AI supports and facilitates the work of system engineers, and whether an AI-based modeling environment is trusted and deemed usable by its users. The study we've conducted provide very encouraging results about usability, and AI-assisted sketching. We acknowledge that providing tips and suggestions to the users alongside an explanation on why this suggestion is well evaluated by the AI, increased both a faster adaptation and an increased confidence in using BOARD-AI.

Data availability statement

The original contributions presented in the study are included in the article/supplementary materials, further inquiries can be directed to the corresponding author.

Author contributions

SC-P has developed and validated the Neural Network (NN) classifier, contributed to the design of the NN training interface and designed the data acquisition process to train the NN classifier; NH has led the writing of the paper and coordinated the different contributions, he has developed the BOARD-AI Human-Machine modeling interface, and integrated the different software modules; AA has modeled the planning problem, integrated the planner, and analyzed the evaluation; MP-S has participated in the experimental design for the assessment of BOARD-AI with final users, and performed the analysis of the assessment.

Conflict of interest

The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

Publisher's note

References

1. Robertson B, Radcliffe D. Impact of CAD tools on creative problem solving in engineering design. *Computer-Aided Des* (2009) 41:136–46. doi:10.1016/j.cad.2008.06.007

2. Rudin C. Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. *Nat Mach Intell* (2019) 1:206–15. doi:10.1038/s42256-019-0048-x

3. Botre R, Sandbhor S. Using interactive workspaces for construction data utilization and coordination. *Int J Construction Eng Management* (2013) 2:62–9.

4. Alblawi A, Nawab M, Alsayyari A. A system engineering approach in orienting traditional engineering towards modern engineering. 2019 IEEE Global Engineering Education Conference (EDUCON). IEEE (2019). p. 1559–67.

5. Lank E, Thorley J, Chen S. An interactive system for recognizing hand drawn UML diagrams. Proceedings of the 2000 conference of the Centre for Advanced Studies on Collaborative research (2000), 7. doi:10.1145/782034.782041

6. Notowidigdo M, Miller RC. *Off-line sketch interpretation*. Arlington, VA: AAAI fall symposium (2004). p. 120–6.

7. Hammond T, Tahuti DR. A geometrical sketch recognition system for UML class diagrams. *SIGGRAPH Courses (ACM)* (2006) 25.

8. Alvarado C, Davis R. *SketchREAD: A multi-domain sketch recognition engine*, 34. San Diego, CA: ACM SIGGRAPH 2007 courses (2007).

9. Paulson B, Hammond T. PaleoSketch: Accurate primitive sketch recognition and beautification. *Proc 13th Int Conf Intell user Inter* (2008) 1–10.

10. Bresler M, Van Phan T, Prusa D, Nakagawa M, Hlavác V. Recognition system for on-line sketched diagrams. 2014 14th International Conference on Frontiers in Handwriting Recognition (2014), 563–8. doi:10.1109/ICFHR.2014.100

11. Bresler M, Prusa D, Hlaváác V. Online recognition of sketched arrow-connected diagrams. *Int J Doc Anal Recognit* (2016) 19:253–67. doi:10.1007/s10032-016-0269-z

12. Vesin B, Jolak R, Chaudron MR. Octouml: An environment for exploratory and collaborative software design. 2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C). IEEE (2017). p. 7–10.

13. Zhang X, Li X, Liu Y, Feng F. A survey on freehand sketch recognition and retrieval. *Image Vis Comput* (2019) 89:67–87. doi:10.1016/j.imavis.2019.06.010

14. Albore A, Hili N. From informal sketches to system engineering models using AI plan recognition: Opportunities and challenges. AAAI 2020 Spring Symposium Series (2020).

15. Hili N, Albore A, Baclet J. From informal sketches to systems engineering models using AI plan recognition. In: Lawless WF, Mittu R, Sofge DA, Shortell T, McDermott T, editors. *Systems engineering and artificial intelligence*. Springer (2021). p. 451–69.

16. Rosenfeld A, Richardson A. Explainability in human-agent systems. *Auton. Agents Multi-Agent Syst.* (2019) 33 (6):673–705. doi:10.1613/jair.2972

17. Richter S, Westphal M. The LAMA planner: Guiding cost-based anytime planning with landmarks. *J Artif Intell Res* (2010) 39:127–77. doi:10.1613/jair.2972

18. Bhowmik S, Sarkar R, Nasipuri M, Doermann D. Text and non-text separation in offline document images: A survey. *Int J Doc Anal Recognit* (2018) 21:1–20. doi:10.1007/s10032-018-0296-z

19. Wüest D, Seyff N, Glinz M. Flexisketch: A mobile sketching tool for software modeling. In: *International conference on mobile computing, applications, and services*. Springer (2012). p. 225–44.

20. MyScript. *Myscript home page* (2020). Available at: https://www.myscript.com/(Accessed 05 13, 2022).

21. Jordan MI, Mitchell TM. Machine learning: Trends, perspectives, and prospects. *Science* (2015) 349:255–60. doi:10.1126/science.aaa8415

22. Jain AK, Mao J, Mohiuddin KM. Artificial neural networks: A tutorial. *Computer* (1996) 29:31–44. doi:10.1109/2.485891

23. Sordo M. Introduction to neural networks in healthcare. In: *Open clinical: Knowledge management for medical care* (2002).

24. McNelis PD. *Neural networks in finance: Gaining predictive edge in the market*. Academic Press (2005).

25. Haddadi F, Khanchi S, Shetabi M, Derhami V. Intrusion detection and attack classification using feed-forward neural network. 2010 Second international conference on computer and network technology. IEEE (2010). p. 262–6.

26. Anderson JA. *An introduction to neural networks*. MIT press (1995).

27. Goldberg Y. Neural network methods for natural language processing. *Synth lectures Hum Lang Tech* (2017) 10:1–309. doi:10.2200/s00762ed1v01y201703hlt037

28. Ghallab M, Nau D, Traverso P. *Automated planning: Theory and practice*. Elsevier (2004).

29. Hollnagel E. Plan recognition in modelling of users. *Reliability Eng Syst Saf* (1988) 22:129–36. doi:10.1016/0951-8320(88)90070-1

30. Kautz HA, Allen JF. Generalized plan recognition. *Proc Fifth AAAI Natl Conf Artif Intelligence* (1986) 86:32–7.

31. Avrahami-Zilberbrand D, Kaminka G, Zarosim H. Fast and complete symbolic plan recognition: Allowing for duration, interleaved execution, and lossy observations. Proc. of the AAAI Workshop on Modeling Others from Observations. Edinburgh, Scotland, United Kingdom: MOO (2005).

32. Fikes RE, Nilsson NJ. Strips: A new approach to the application of theorem proving to problem solving. *Artif intelligence* (1971) 2:189–208. doi:10.1016/0004-3702(71)90010-5

33. Hansen EA, Zhou R. Anytime heuristic search. *J Artif Intell Res* (2007) 28:267–97. doi:10.1613/jair.2096

34. Thayer JT, Ruml W. Faster than Weighted A*: An optimistic approach to bounded suboptimal search. *ICAPS* (2008) 355–62.

35. Likhachev M, Ferguson D, Gordon G, Stentz A, Thrun S. Anytime search in dynamic graphs. *Artif Intelligence* (2008) 172:1613–43. doi:10.1016/j.artint.2007.11.009

36. Bhatia A, Svegliato J, Zilberstein S. On the benefits of randomly adjusting anytime weighted a. *Proc Int Symp Comb Search* (2021) 12:116–20.

37. Ramírez M, Geffner H. Plan recognition as planning. In: *Twenty-first international joint conference on artificial intelligence* (2009).

38. McDermott D, Ghallab M, Howe A, Knoblock C, Ram A, Veloso M, et al. *PDDL-the planning domain definition language* (1998).

39. Edelkamp S, Hoffmann J. *PDDL2.2: The language for the classical part of the 4th international planning competition*. Tech. rep.. Freiburg im Breisgau, Germany: University of Freiburg (2004). p. 195.

40. Thiébaux S, Hoffmann J, Nebel B. In defense of PDDL axioms. *Artif Intelligence* (2005) 168:38–69. doi:10.1016/j.artint.2005.05.004

41. Helmert M. The fast downward planning system. *J Artif Intell Res* (2006) 26:191–246. doi:10.1613/jair.1705

42. MDN Web Docs. Pointer events. Available at: https://developer.mozilla.org/en-US/docs/Web/API/Pointer_events (2021). Accessed: 2021-03-31.

43. Nair V, Hinton GE. In: Fürnkranz J Joachims T, editors. *Rectified linear units improve restricted Boltzmann machines*. Haifa, Israel: ICML (Omnipress) (2010). p. 807–14.

44. Glorot X, Bordes A, Bengio Y. Deep sparse rectifier neural networks. In: Gordon G, Dunson D, Dudík M, editors. *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, 15. Fort Lauderdale, FL, USA: PMLR (2011). p. 315–23.

45. Brooke J. SUS: A quick and dirty usability scale. *Usability Eval industry* (1996) 189.

46. Jolak R, Vesin B, Isaksson M, Chaudron MR. Towards a new generation of software design environments: Supporting the use of informal and formal notations with octouml. In: *HuFaMo@ MoDELS* (2016). p. 3–10.

47. Hili N, Farail P. BabyMOD, a collaborative model editor for mastering model complexity in MBSE. International Workshop on Model-Based Space Systems and Software Engineering (2020), 1–4.