Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.

*Digital Object Identifier 10.1109/ACCESS.2022.Doi Number*

# Transforming Sketches of UML Use Case Diagrams to Models

**Mantas Ražinskas[1], Benas Miliūnas[1], Mantas Jurgelaitis[1], Lina Čeponienė[1], and Lina Bisikirskienė[1]**

[1]Department of Information Systems, Faculty of Informatics, Kaunas University of Technology, 51368 Kaunas, Lithuania

Corresponding author: Mantas Ražinskas (mantas.razinskas@ktu.edu).

**ABSTRACT** Considering that use case specification often plays the most important role in requirements engineering, transitioning from a design on paper to an editable, standard-compliant model is a fitting candidate for automation. This paper presents a methodology for transforming Use Case diagram sketches into standardized UML Use Case models. The methodology utilizes a tailored Faster R-CNN network for detecting objects in the sketch, combined with the Google Cloud Vision API for text recognition. It also includes techniques for detecting and identifying diagram relations, their direction, start, and end points, and integrating these elements into a structured UML diagram. The final output is transformed into an XML Metadata Interchange (XMI) format file, ensuring compatibility with various UML Computer-Aided Software Engineering (CASE) tools. The methodology was implemented in a tool and empirically validated through experiments using a dataset of 32 Use Case diagram sketches, made up of 765 instances of Use Case diagram elements. The tool's performance demonstrates 94% accuracy in symbol recognition and overall diagram digitization accuracy of 95%.

**INDEX TERMS** diagram digitization, image processing, requirements engineering, sketch recognition, UML, Use Case diagrams, XMI.

## I. INTRODUCTION

In software system development, Unified Modeling Language (UML) [1] diagrams play an indispensable role in facilitating the transition from conceptual ideas to their implementation. By providing a standardized visual language, UML diagrams enable analysts, architects, developers, and stakeholders to articulate complex system requirements, architectures, or interactions without the need to delve into the specifics of implementation technologies [2] [3]. This is particularly prevalent in the requirements engineering phase, which encompasses the elicitation, documentation, and validation of system requirements [4]. UML Use Case diagrams are a valuable tool during this phase, providing a high-level view of system functionalities and interactions from the user's perspective [5] [6] [7]. Use Case diagrams facilitate communication among stakeholders and serve as a foundation for a detailed system design and implementation. Initially, sketches of Use Case diagrams are often sufficient to enable discussions and capture initial ideas during elicitation. However, as the project progresses, these sketches need to be transformed into models to ensure consistency and completeness in the requirements specification. Models not only ensure adherence to standard UML notation but also enable diagram verification [8] [9] [10]. The verified model can then be used for comprehensive system documentation, enabling further system support and development.

To create the initial Use Case diagram sketches, various digital tools and collaborative media from paper to whiteboards can be used. Although these tools offer a convenient platform for the informal creation of diagrams during the initial stages of requirements engineering, they also introduce challenges in transitioning these preliminary designs to more formalized digital representations that are conducive to further development and detailed analysis. The main challenge lies in the informal and often ad-hoc manner in which these tools are used during the brainstorming and preliminary design phases. The UML diagrams created in this context, although quickly and collaboratively produced, may be marked by inconsistencies, inaccuracies, and a lack of adherence to the UML standard, necessitating a time-consuming manual process to transition them to models. The manual digitalization of sketches can lead to a loss of information and/or design decisions captured in them [11].

On the other hand, Computer-Aided Software Engineering (CASE) tools enable the development of models rather than just plain drawings. These tools facilitate the development of UML models that are not only visual representations but encompass a set of model elements that can capture system requirements and design. The elements of the model can be used to create diagrams that represent the system under development from various viewpoints, thus enhancing the clarity and precision of the design process [2]. CASE tools also provide additional functionalities, such as model consistency checking, version control, and automated documentation generation, which contribute to the overall quality and maintainability of the system models. Therefore, the transition from informal sketches to models plays an important role in ensuring that initial conceptualizations are translated into detailed and implementable specifications.

Consequently, main aim of our research is to facilitate requirements engineering process by proposing an approach for transforming informal sketches of UML Use Case diagrams to UML Use Case models. These models not only ensure adherence to the UML standard but can also be further developed using the UML CASE tool. Therefore, we have developed a methodology, which encompasses components for identification of sketch objects (based on tailored Faster R-CNN network), text recognition (using Google Cloud Vision API), detection of direction, start and end for relations in the diagram, linking all the detected elements into a complete model and transforming this model into XML Metadata Interchange (XMI) format file. The use of the XMI format allows for standardized UML model data interchange among the supporting CASE tools, thus maintaining the integrity and usability of UML diagrams across different software environments [12].

The proposed methodology was implemented as a tool that transforms the provided image into an XMI file. The tool was used during the experimental evaluation of the methodology, which encompassed the evaluation of both the recognition of elements and overall diagram digitization, where 95% accuracy was achieved.

Existing approaches in the area of diagram sketch recognition usually focus on other diagram types like Class diagrams [14] [15] or require manual intervention to correctly transform sketches to digital models, resulting in inefficiencies [16], [17]. Our proposed methodology overcomes these limitations by handling the process of transforming sketches of UML Use Case diagrams into XMI format model, compatible with CASE tools and enabling model validation, verification, and further development.

The rest of the paper is structured as follows. In the Related Work section, research in the area of diagram sketch digitization is presented. In the third section, the proposed methodology is discussed, detailing the approach for digitizing sketches of UML Use Case diagrams into UML models, including the use of machine learning algorithms, image processing techniques, and the generation of XMI files. The fourth section discusses the implementation of the transformation tool. In the Results section, an experimental evaluation of the proposed methodology is presented, including experiments conducted to assess the accuracy of the digitization process. The sixth section analyzes the results and discusses the performance of the solution, its limitations, and potential areas for improvement and future research perspectives.

## II. RELATED WORK

Research on diagram sketch recognition has mostly been directed towards two major categories: online-targeted diagram recognition [18] [19] [20] and offline-targeted (sketches) recognition and digitization [21] [22]. In the first category, UML diagrams drawn using smartboards are recognized in real time and digitized based on information regarding drawing actions performed by the user. This category of sketch digitization deals with optimization problems that aim to group the closest UML diagram symbols in space and time [23]. On the other hand, the offline-targeted recognition category concentrates on digitizing diagram sketches when a complete image is available, but there are no temporal and spatial data on the drawn elements. In this area, machine learning and computer vision techniques [21] [24] [25] are commonly used to recognize UML notation symbols and then connect them to reconstruct the full structure of the diagram. In addition to UML symbol recognition and diagram generation, image processing, text recognition, and diagram classification algorithms have been used [26] [25] [27] [15] [20] [13].

In the area of sketch digitization, the recognition of various types of diagrams is analyzed, including UML Class diagrams [14], State machines or finite automata [26], BPMN diagrams [22], and Flowcharts and mind maps [29]. Recognition of UML Use Case diagrams has not been widely analyzed. One approach to using Use Case sketch recognition is presented in [18], where a parsing strategy for recognizing hand-drawn diagrams using sketch grammars is presented. However, this research focuses on the recognition of real-time drawn diagrams using whiteboards (online-targeted recognition) and does not analyze the process of transformation from complete sketch to UML model.

The final result of the sketch digitization process varies in different approaches, but in the area of UML diagram recognition, the recognized diagrams have the highest value if they are transformed into the XMI format, which enables model information interchange among various UML CASE tools. The transformations presented in [14] and [30] provide the recognized result in XMI format, but they concentrate on other UML diagram types and do not cover UML Use Case diagram recognition.

Neural networks based on Convolutional Neural Network (CNN) architectures are widely used for image recognition problems [21] [31] [32]. One of such approach is DrawNet, a deep neural network based on CNN keypoint detection, which is capable of successfully recognizing individual diagram elements [21]. DrawNet achieved high accuracy across multiple benchmark datasets, but the accuracy of individual class recognition was significantly higher than the overall diagram recognition accuracy. This indicates that the most challenging task in the digitization of diagrams from sketches is not the recognition of individual diagram symbols but the ability to connect and reconstruct them into a final diagram. While DrawNet concentrates on flowchart diagram recognition, the findings of this research are valuable in the area of Use Case diagram recognition as well.

Another CNN-based approach, Arrow R-CNN [32] is a deep neural network based on the Faster R-CNN neural network, which in turn is derived from the R-CNN network. The operation of the Arrow R-CNN can be broken down into the following main steps: the CNN base network (ResNet-101) generates feature vectors from a given image. The RPN network forms a set of Regions of Interest (RoI) based on feature vectors. The RoI proposal network classifies an object and proposes a bounding box. Additionally, key points for the heads and tails of the detected arrows are formed. The final processing model integrated the detected elements into a common structure. The overall model accuracy was measured based on the number of correctly recognized diagrams in the test sample. A correctly recognized diagram is considered one in which all diagram symbols are identified and connected correctly. The Arrow R-CNN model demonstrated high accuracy in recognizing individual diagram classes, whereas the diagram recognition accuracy reached 85%. Similar to the Arrow R-CNN approach, we used Faster R-CNN as a basis for the sketch object recognition component with a training configuration based on UML Use Case diagrams.

Alternative object detection algorithms like YOLO (You Only Look Once) [33] [34] [35] [36] and SSD (Single Shot MultiBox Detector) [37] [38] [39] are widely used in various applications due to their speed and efficiency in real-time object detection. YOLO treats object detection as a single regression problem, predicting bounding boxes and class probabilities directly from the image in one evaluation, making it fast but potentially less accurate for complex symbol recognition tasks. SSD balances speed and accuracy by applying convolutional filters to different feature maps. The choice of algorithm ultimately depends on the specific usage and requirements of the application. Several papers on comparative analysis of YOLO, SSD, and Faster R-CNN reveal their distinct strengths [40] [41] [42]. In comparisons, Faster R-CNN excels in detection accuracy, albeit at a slower pace, and is suitable for scenarios prioritizing precision. In our approach, Faster R-CNN provides accuracy for recognizing complex diagram elements, which is more important than speed [43] in the context of UML Use Case diagram sketch recognition.

Another research, that analyzed sketch recognition was Sketch2Process [22], an end-to-end sketch recognition approach specifically designed for BPMN process models. Sketch2Process utilizes a neural-network-based architecture and achieves high accuracy in recognizing the complex elements of BPMN models, thereby facilitating the digitalization of process sketches drawn by hand. The creation of a dataset consisting of 704 manually annotated BPMN models ensured the robustness and scalability of the approach. By outperforming existing state-of-the-art solutions, Sketch2Process demonstrates the potential of applying advanced machine-learning techniques to the domain of process modeling, offering a practical tool for transforming informal sketches into digital representations suitable for further processing and analysis.

Another approach to sketch recognition is Flowmind2Digital [29], an end-to-end recognition and conversion method for hand-drawn flowcharts and mind maps, collectively referred to as flowminds. This approach utilizes a neural network architecture and keypoint detection technology to enhance the recognition accuracy. The hdFlowmind dataset, consisting of 1776 hand-drawn and manually annotated flowminds, covers a wide range of scenarios, demonstrating improvements over previous state-of-the-art methods. Flowmind2Digital facilitates the automated conversion of hand-drawn flowminds into editable digital formats suitable for use in software, such as Microsoft PowerPoint and Visio.

The Img2UML tool [14] analyzes the extraction of UML models from the images. This tool converts UML Class models from image formats into XMI files, which can be imported into CASE tools. The ability to export UML models into XMI format ensures interoperability and facilitates further analysis. This study relates to our research in terms of transforming an informal sketch into a standardized representation of the UML model in XMI format, but the Img2UML approach is specialized for Class diagrams, whereas ours concentrates on Use Case diagrams.

Recently, LLMs (Large Language Models) have been improved in such a way, that they theoretically can analyze the image of UML diagram [16] and transform it to XMI file. Although the idea of using LLM for UML recognition of diagrams sketches seems promising, current research shows that there is a need to keep the human in the transformation loop to overcome the limitations of the LLM approach [16], [17] as the LLM output can be unreliable, often requiring expert intervention and editing.

Together, these studies contribute to the developing field of diagram generation from sketches and textual descriptions, employing a range of computational techniques, from graph transformations to deep learning [25]. In Table I, a comparison of different methodologies for diagram digitization is provided. It is important to note that direct comparisons of efficiency between these methodologies cannot be made, as they target different types of diagrams, and their recognition tasks involve distinct notations and processing challenges. While accuracy criteria in the table outlines the results of other proposed approaches, a direct comparison cannot be made considering that the approaches are intended for different diagram types, use different data sets, the outlined methods have different number of classes and features that they need to consider.

Table I. COMPARISON OF DIAGRAM DIGITIZATION METHODOLOGIES

| Feature | Our Methodology | Arrow R-CNN [32] | DrawNet [21] | Sketch2Process [22] | FlowMind2Digital [29] | Img2UML [14] |
|---|---|---|---|---|---|---|
| Diagram Type | UML Use Case | Flowcharts, Finite automata | Flowcharts | BPMN | Flowcharts, Mind maps | UML Class |
| XMI support | Yes | No | No | No | No | Yes |
| Text Recognition Support | Yes | Yes | Yes | Yes | Yes | Yes |
| Accuracy | 95 | 85 | 85 | 89 | 87 | 89 |
| Machine Learning Method | CNN | CNN | CNN | NN | NN | N/A |

Existing diagram digitization approaches achieve high symbol recognition accuracy; however, these solutions are primarily designed for the digitization of flowcharts, process models, or UML Class diagrams, but do not support the recognition of UML Use Case diagrams. Our research addresses these gaps by proposing a methodology and implementing a tool specifically tailored for the digitization of UML Use Case diagrams, which supports the recognition of UML Use Case model elements and their relationships, ensuring the reconstruction of the original sketch structure. Moreover, our tool enables the export of digitized diagrams to XMI format, facilitating integration with other CASE tools for further development.

## III. METHODOLOGY

This section outlines the methodology of transforming preliminary Use Case diagram sketches into UML models in the XMI format. The main components of the proposed methodology are shown in Fig. 1.

### *A. Methodology Components Overview*

The Detection component enables the identification of regions from the sketches, analyzes these regions to determine connection types, and removes redundant regions. The OCR component uses Google Cloud Vision API to detect and interpret text to label diagram elements. The Keypoint detection component is required to identify the connection points of the sketch elements, thus enabling the detection of connections between the UML elements. The Connector component couples the detected sketch regions with their names and joins the connections between diagram elements and symbols. The XMI Converter component converts the processed data into an XMI file accepted by UML CASE tools, allowing further manipulation, extension, and use of the UML Use Case diagrams.

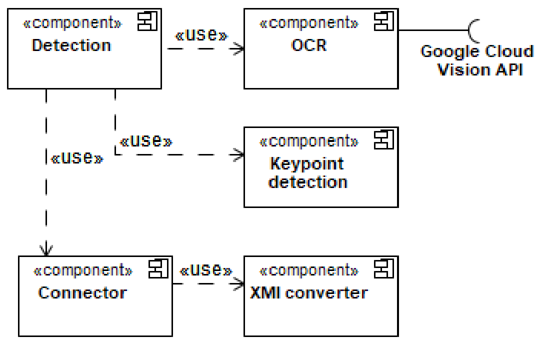


FIGURE 1. The main components used for digitization of UML diagram sketches

For a detailed representation of the proposed methodology, the activity diagram in Fig. 2 depicts the process of digitizing diagram sketches into UML diagrams. The metamodel of the analyzed elements of the sketch is presented in more detail in Fig. 5, while the metamodel of analyzed UML Use Case diagram elements is presented in Fig. 6.

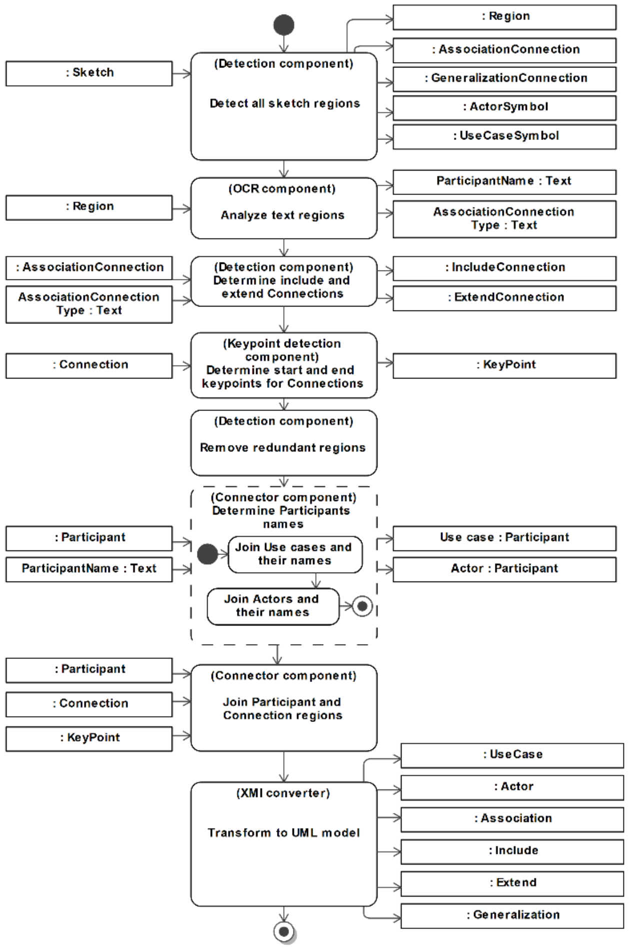

: Sketch
(Detection component) Detect all sketch regions
: Region
: AssociationConnection
: GeneralizationConnection
: ActorSymbol
: UseCaseSymbol
: Region
(OCR component) Analyze text regions
ParticipantName : Text
AssociationConnection Type : Text
: AssociationConnection
AssociationConnection Type : Text
(Detection component) Determine include and extend Connections
: IncludeConnection
: ExtendConnection
: Connection
(Keypoint detection component) Determine start and end keypoints for Connections
: KeyPoint
(Detection component) Remove redundant regions
(Connector component) Determine Participants names
: Participant
ParticipantName : Text
Join Use cases and their names
Join Actors and their names
Use case : Participant
Actor : Participant
: Participant
: Connection
: KeyPoint
(Connector component) Join Participant and Connection regions
(XMI converter) Transform to UML model
: UseCase
: Actor
: Association
: Include
: Extend
: Generalization

FIGURE 2. The process of transforming diagram sketch to UML model

### B. Object Detection Using Faster R-CNN

First, the detection component aims to identify all Use Case diagram symbols in a given sketch and return a set of Regions. The TensorFlow Object Detection 2 library implementation of the Faster R-CNN network was used to implement the component with a training configuration based on UML 2.5 Use Case diagram symbols, and context. The symbols identifying the components are UseCases, Actors, Associations, Include and Extend Relationships and Generalizations (as presented in Fig. 5). The input to this component is a diagram Sketch that is resized so that the longer side is equal to 1,333 pixels, which is the standard format required by the Faster R-CNN ResNet neural network used in this step.

To train the diagram symbol recognition model, a new dataset of Use Case diagrams was created. The dataset encompasses a total of 346 diagram sketches, 288 of which (83.2%) are whiteboard or analogous tool-drawn diagram images and 58 (16.7%) are hand-drawn diagrams. Various sizes and positions of elements were created in the dataset to achieve better model accuracy. In total, the dataset included 1907 Association Relationship elements (36.6%), 1803 Use Case elements (34.6%), 838 Actor elements (16.1%), and 658 Generalization Relationship elements (12.7%). Association Relationship elements consist of non-directional Association Relationships and Directed Relationships before preprocessing. The TensorFlow Object Detection API 2 software interface was used to access the pretrained models. To optimize model performance and training time, transfer learning methods were applied to the weights and other model parameters. The Faster R-CNN ResNet152 V1 800 × 1333 neural network was selected as the base architecture, which was also used in other analyzed approaches. The Faster R-CNN model was configured for training according to UML 2.5 Use Case diagram symbols, and context. Table II presents the key hyperparameters used.

**Table II. HYPERPARAMETERS USED IN FASTER R-CNN MODEL TRAINING FOR UML USE CASE DIAGRAM RECOGNITION**

| **Hyperparameter** | **Value** |
|---|---|
| **Learning Rate (Base)** | 0.04 |
| **Num_classes** | 5 |
| **Batch Size** | 4 |
| **Total Steps** | 5000 |
| **Momentum** | 0.9 |
| **IoU Threshold** | 0.6 |

After detecting the objects, the output is a set of Regions *R*, where each Region is defined by three values (b, c, and s). The value b is the bounding box of the object, c is the class describing the object (label), and the value s is a scalar that determines the reliability of the prediction of value c. An example of the detection results is shown in Fig. 3.

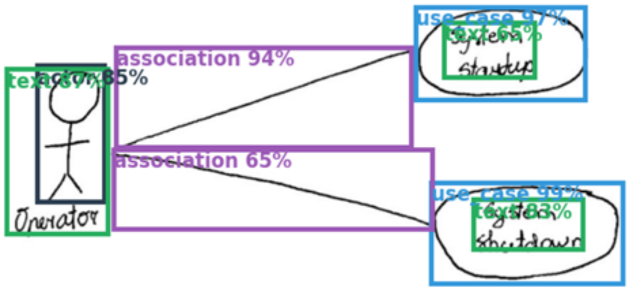
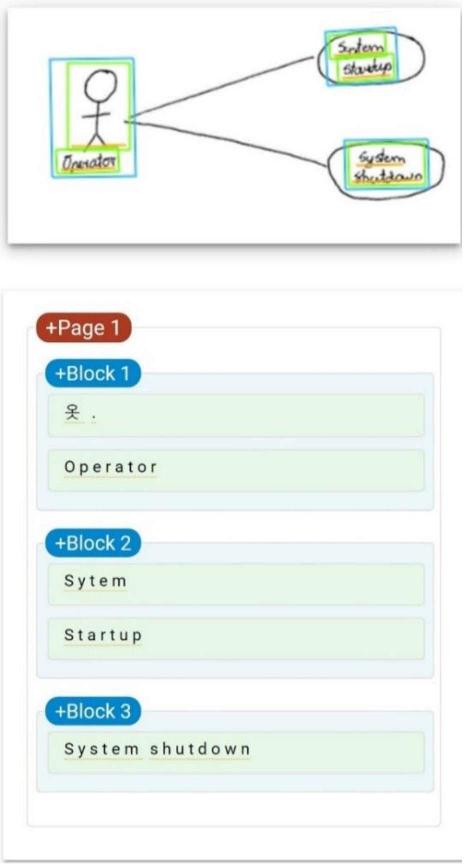


**FIGURE 3.** Result of detection of regions

### C. Text Recognition Using Google Cloud Vision API

The OCR component performs text recognition on a classified Text Region (as shown in Fig. 4). Each Text Region (cut out part of the image) was submitted to Google Cloud Vision API. Given a raw image as the input, the service returns a set of text strings, where each string consists of a sequence of words. The strings and sequences of words are concatenated into a single text line, and the resulting text is associated with a Text Region.



```
웃 .
Operator
Sytem
Startup
System shutdown
```

**FIGURE 4.** Results of Google Cloud Vision OCR text recognition on a classified text region

### D. Detection of Diagram Relations

A component to determine the direction and orientation of the Directed Relationship is also required. The component specifies the coordinates of the start and end KeyPoints of the Association for the classified Regions of the Association. For a nondirectional AssociationConnection, it does not matter which of the points is the start or end because this Connection only states that the symbols are related. For Generalization, Include, and Extend Directed Relationships, it is important to correctly determine not only the coordinates of the KeyPoints but also which of these points is the beginning and which is the end. Otherwise, symbols will be mapped incorrectly. For example, incorrectly setting the start and end points of a GeneralizationConnection between two actors will result in an incorrect inherited functionality between the actors. In a Sketch, IncludeConnections and ExtendConnections must have arrowheads. In order to determine which of the KeyPoints is the start point and which is the end point, an algorithm is applied in which KeyPoint D1 is selected, and it is checked whether there is an arrowhead Region in its position. If there is an arrowhead Region at the position of KeyPoint D1, then this point is considered the end point, and KeyPoint D2 is set as the starting point. Otherwise, D1 is set as the starting point and D2 is set as the endpoint. Arrowhead Regions may not always be recognized. If no head Region exists at any of the points D1 or D2, the Histogram Projection Method (HPM) is used. Around each Keypoint, HPM is performed in a small radius to determine which of the points has more foreground pixels. The point that has more foreground pixels is considered the endpoint (this point contains the arrowhead region).

### E. Model Reconstruction and XMI Conversion

After detecting all Sketch Regions, analyzing Text Regions, and determining IncludeConnections and ExtendConnections, while also obtaining start and end KeyPoints for Connections, redundant Regions are identified and removed. Furthermore, there is also a Connector component (Fig. 1) that links individual diagram elements to a common structure.

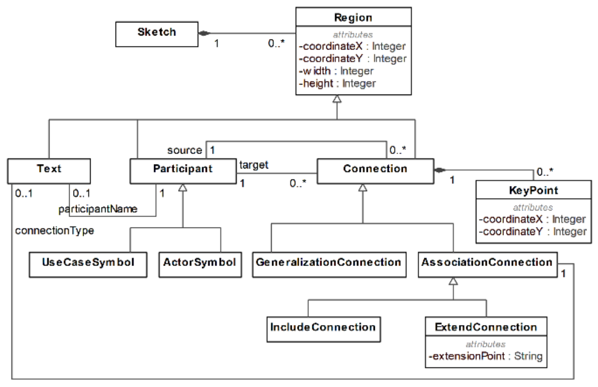


FIGURE 5. Metamodel of analyzed elements of the diagram sketch

Symbols in the diagram are connected to each other based on the assumption that the closer the elements are to each other, the more likely they are to be related. Thus, the elements are connected to each other according to the distance, which is calculated using the Euclidean distance formula. In addition, the available knowledge about the rules for creating a Use Case diagram is used.
- Connections cannot be linked to themselves.
- Only Participants of the same type can be connected by GeneralizationConnection.
- Only Use Cases can be linked by AssociationConnection
- AssociationConnections can have <<include>> or <<extend>> tags.
- The name of the UseCaseSymbol is in the center of the bounding box of UseCaseSymbol.
- The name of the ActorSymbol is usually under ActorSymbol.

The main input consists of recognized Regions, where each one has a name, x and y coordinates, and width and height. The Association Regions additionally have start and end Keypoints. The result is the information of individual Regions in JSON format, which defines the connected Regions into a common model, forming a UML Use Case diagram. When detailing the mapping algorithm used by the Connector component, the process can be divided into separate stages. First, the algorithm determines the names of the Use Cases, iterates through each Use Case Region, and checks whether there is a Text Region in the center of the Use Case region. If the Text Region is present, it is merged with the Use Cases Region and becomes the name of the Use Case. If no Text Region exists, an empty name is used for the Use Case. Next, the Actor names are determined by iterating through each Actor Region and finding the closest Text Region below the Actor. If the Text Region is found, the Actor name is determined based on the Text Region; otherwise, the Actor name is left blank. The next step in the diagram component-mapping algorithm is to assign Actors to the ends of the Generalization Relationship by iterating through each GeneralizationConnection Region. The starting point of the Generalization Relationship is selected and assigned to the nearest Actor. Next, the endpoint of the Connection is selected, and the Actor closest to the point is found, which is assigned to the other end of the Connection. The Actor assigned to the starting point inherits the functionality of the Actor assigned to the end point. If at least one of the ends of the Connection cannot be assigned to an Actor, then such a GeneralizationConnection is rejected and not processed further. Next, the mapping of AssociationConnections to Use Cases is performed by iterating through each Region of the directional AssociationConnections, and the starting point is selected and assigned to the closest Use Case. The same steps are repeated for the endpoint of the connection next to the Connection of the nearest Text Region whose content contains the stereotype <<include>> or <<extend>>.

According to this Text Region, it is decided whether AssociationConnection is an ExtendConnection or IncludeConnection. If the Connection is an ExtendConnection, Text Regions next to or on the Connection that defines the condition of the extension are additionally searched. The text of the extend clause is usually enclosed in parentheses, and if such a Text Region is found, an ExtendConnection's extensionPoint is created based on it. Similarly, if no Text Region defining the Include subtype is found near the IncludeConnection Region, IncludeConnection is converted to a nondirectional AssociationConnection. After the ExtendConnections and IncludeConnections are processed, AssociationConnections between Use Cases and Actors, or two Actors or Use Cases, are finally mapped by iterating through the AssociationConnections and finding the nearest Use Case or Actor Region for each end of the Connection and assigning it. Finally, the multiplicity of the nondirectional AssociationConnections is determined. The ends of each Connection are searched for a Text Region that has the specified multiplicity, iterating through all AssociationConnections Regions and finding the closest Text Region that defines the multiplicity. The identified Text Region is assigned to the end of the AssociationConnection.

After completing all the steps of the algorithm, a final structure that will correspond to the metamodel of the Use Case diagram (Fig. 6) is formed from which an XMI file can be generated. The latter can then be imported into a UML CASE tool that supports this format for further model modification.

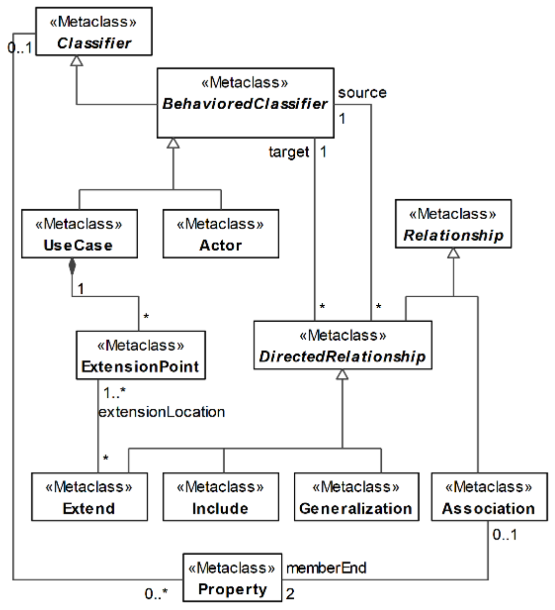


FIGURE 6. Fragment of UML metamodel depicting Use Case diagram metaclasses

Fig. 7 shows a simple example of how a hand-drawn sketch became a UML Use Case diagram using our methodology. In the sketch, all Use Cases, Actors Association, and Directed and Generalization Relationships are digitized in the correct directions with the correctly identified elements.

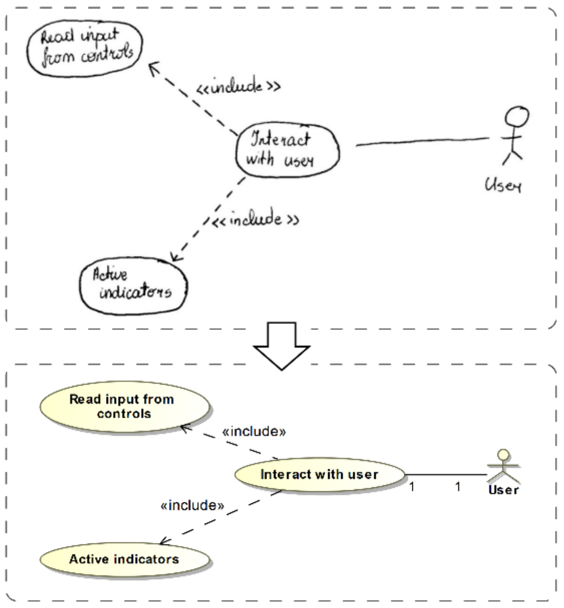


FIGURE 7. An example of a hand-drawn sketch and the resulting UML Use Case diagram

## IV. Implementation of the proposed methodology

For the proposed methodology, we have implemented the Use Case digitization tool S2D (the source code of the tool is available at the GitHub repository: https://github.com/benasmi/s2d). Users can use the tool by uploading a sketch for digitization, controlling the digitization process, and choosing subsequent actions upon reviewing the digitized output. The tool itself automates the transformation of the uploaded sketch into a structured UML Use Case diagram, ensuring validation against the UML 2.5 specification. Upon successful digitization, the system offers the user several post-processing options, including the ability to view or download an XMI file that can later be loaded into a UML CASE tool (Fig. 8).

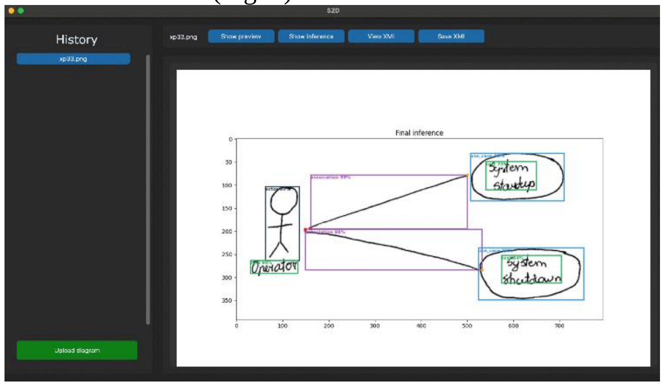


FIGURE 8. A screenshot of the tool after successfully digitizing the sketch

The tool utilizes a tailored model for diagram symbol detection purposes. For the implementation of the tool, a monolithic architecture was used, consolidating the user interface and backend processes within a single deployment unit, thereby streamlining deployment and maintenance operations.

## V. Experimental results

The implemented Use Case digitization tool based on the proposed methodology was evaluated in two experiments. In the first experiment, an evaluation of sketch diagram symbol detection and recognition accuracy were performed, and in the second part, the overall accuracy of Use Case diagram digitization was evaluated.

For both experiments, a new dataset of 32 diagrams was compiled, which consisted of Use Case diagram sketches, 15 of which were drawn by hand on paper, and the other 17 of which were specified using various tools, such as draw.io, Visual Paradigm, and Miro. The diagrams were specified according to the UML 2.5 specification (the dataset and calculations are available at the GitHub repository: https://github.com/benasmi/s2d/tree/main/detection/xp).

In total, the 32-diagram set is composed of 765 elements: 39.1 percent of the elements is written text, 29.4 percent is relationships, 21.7 percent is Use Cases, and 9.8 percent is actors (as presented in Table III).

Table III. COMPOSITION OF THE 32-DIAGRAM DATASET USED FOR EVALUATING THE USE CASE DIGITIZATION TOOL

| Class | Occurrences |
|---|---|
| **Text** | 299 |
| **Relationships** | 225 |
| **Use Cases** | 166 |
| **Actors** | 75 |

### F. *Experimental Setup*

For both experiments, each diagram was digitized using the implemented tool S2D. Each diagram element accuracy was calculated by first determining the True Positive (TP), False Negative (FN), and False Positive (FP) results. TP occurrences were noted when the diagram element was identified correctly, FN when an occurrence was missed, and FP when an occurrence was determined, which was not specified in the diagram.

Using these three result values, the recall (Rec.), and precision (Prec.). metrics were calculated for each class and later for each diagram, where the same metrics were also calculated by micro-averaging (Micro) and macro-averaging (Macro) for each class results.

### G. *Diagram symbol recognition experiment*

During the experiment, each sketch was passed through the developed diagram symbol recognition model. Each sketch was then evaluated, and recall and precision were calculated for Relationship, Generalization, Actor, Use Case and Text classes.

After each diagram was evaluated, the macro- and micro-averages for each class were calculated as well. Overall, 742 out of 763 elements were identified correctly. Most notably, out of all relationships that include association, Include and Extend directed relationships, 9 FN relations were not recognized, 32 FP were also identified, and 177 TP were identified, which resulted in a lower precision of 0.85. In terms of generalization recognition, 35 instances were identified correctly, and 4 FN and 4 FP were also recognized, resulting in a recall of 0.90 and precision of 0.90. Both Actors and Use Case elements were identified correctly, and only 8 FP were identified in the Use Cases. Finally, text recognition was the lowest among all classes, but recall, like the other classes, was high at 0.97, as 289 text elements were correctly identified, 10 were not found, and 51 were misidentified.

As shown in Table IV, the overall recall is quite high ranging from 0.90 to 1 for all the elements. Unfortunately, the precision suffers as the diagram symbol recognition model tends to overidentify some classes, resulting in precision ranging from 0.85 to 1, the total accuracy (F1 score) is 0.94 when accounting both the recall and precision. Considering that the main output of this step is to identify as many existing elements as possible, the over-identified elements are removed during the postprocessing steps.

Table IV. PERFORMANCE EVALUATION OF THE DIAGRAM SYMBOL RECOGNITION

| Class/Metric | Objects | TP | FN | FP | Rec. | Prec. |
|---|---|---|---|---|---|---|
| **Relationship** | 186 | 177 | 9 | 32 | 0.95 | 0.85 |
| **Generalization** | 39 | 35 | 4 | 4 | 0.90 | 0.90 |
| **Actor** | 75 | 75 | 0 | 0 | 1.00 | 1.00 |
| **Use Case** | 166 | 166 | 0 | 8 | 1.00 | 0.95 |
| **Text** | 299 | 289 | 10 | 51 | 0.97 | 0.85 |
| ***Totals*** | **Macro** | | **Micro** | | **F1 score** | |
| | **Rec.** | **Prec.** | **Rec.** | **Prec.** | | |
| **All** | 0.96 | 0.91 | 0.97 | 0.89 | 0.94 | |

### H. *Diagram digitization experiment*

Besides the diagram symbol recognition, digitization also consists of many additional steps, such as relationship key point detection, element name detection, removal of redundant elements, diagram element relationship establishment, directed relationship refinement, and serialization to the XMI format. Therefore, an additional experiment was also performed which evaluate the overall application of the full-diagram sketch digitization process.

As shown in Table V, during the experiment, the same dataset of diagram sketches was used. The previously recognized elements of the diagrams were also analyzed and postprocessed accordingly. At the end, the generated XMI file was imported to the Magic Systems of Systems Architect CASE tool and compared to the initial Use Case diagram sketch.

It is important to note that during the experiment, the text was considered as correctly digitized within the margin of error of 1 to 4 symbols.

In the same way as previously mentioned, the accuracy of sketch digitization was calculated, but this time, additional Include and Extend directed relationships were identified and included in the element count.

Overall, the precision results have improved and range from 0.92 to 1. A total of 125 Association relationship instances were correctly digitized, 9 Associations were not identified, and 9 Associations were over-identified, resulting in additional digitized Associations. This has resulted in a precision of 0.93, and a recall of 0.93. Additionally, in terms of Use Cases, 1 instance of Use Case was over-identified, resulting in an additional Use Case.

Out of all Extend relations, 4 instances were missed entirely, resulting in a recall of 0.85 for the Extend class, while Include relations were identified correctly in all instances. 6 instances of Generalization were missed as well, resulting in a recall of 0.85 for Generalization. Lastly, the recall of text is at 0.89, while the precision is at 0.99. The lower recall value is due to the additional postprocessing steps, as during postprocessing, the text regions can be divided into several regions. During digitization, these regions are not grouped back together, resulting in text which has missing words, which is then considered as an error.

When compared to the previous experiment, the digitization precision and F1 score are higher, which is due to the implemented postprocessing steps.

Table V. PERFORMANCE EVALUATION OF FULL DIAGRAM SKETCH DIGITIZATION PROCESS

| Class/Metric | Objects | TP | FN | FP | Rec. | Prec. |
|---|---|---|---|---|---|---|
| **Association** | 134 | 125 | 9 | 9 | 0.93 | 0.93 |
| **Extend** | 26 | 22 | 4 | 2 | 0.85 | 0.92 |
| **Include** | 26 | 26 | 0 | 0 | 1.00 | 1.00 |
| **Generalization** | 39 | 33 | 6 | 1 | 0.85 | 0.97 |
| **Actor** | 75 | 75 | 0 | 0 | 1.00 | 1.00 |
| **Use Case** | 166 | 166 | 0 | 1 | 1.00 | 0.99 |
| **Text** | 297 | 267 | 32 | 2 | 0.89 | 0.99 |
| ***Totals*** | **Macro** | | **Micro** | | **F1 score** | |
| | **Rec.** | **Prec.** | **Rec.** | **Prec.** | | |
| **All** | 0.93 | 0.97 | 0.93 | 0.98 | 0.95 | |

To conclude, individual elements are digitized with relatively high recall (0.93) and precision (0.97), matching with the analyzed DrawNet and Arrow R-CNN solutions, where the symbol recognition accuracy reaches 98.4%–99.5%. In obtained results of our experiment, such a recall metric means that 93% of the elements were digitized correctly, while 97% precision means that only those elements that existed in the diagram sketch were digitized. 3% of all digitized elements were additionally identified that did not exist in the sketch.

On the other hand, when evaluating the relationship connections and direction, the recall of such symbol connections is only 0.84 with a precision of 0.95, resulting in an overall F1 accuracy of 0.89 (as shown in Table VI). This means that 16% of the elements in the diagram were not connected or connected with the wrong elements or in the wrong direction. The recall and precision of the connections are mainly determined by the keypoint detection component, improvement of which would leads to a better result of connecting the elements.

Table VI. RELATIONSHIP ACCURACY IN DIAGRAM RECONSTRUCTION

| Class/Metric | Objects | TP | FN | FP | Rec. | Prec. |
|---|---|---|---|---|---|---|
| **Association** | 134 | 125 | 9 | 8 | 0.93 | 0.94 |
| **Extend** | 26 | 16 | 10 | 3 | 0.62 | 0.84 |
| **Include** | 26 | 26 | 0 | 0 | 1.00 | 1.00 |
| **Generalization** | 39 | 32 | 7 | 0 | 0.82 | 1.00 |
| ***Totals*** | **Macro** | | **Micro** | | **F1 score** | |
| | **Rec.** | **Prec.** | **Rec.** | **Prec.** | | |
| **All** | 0.84 | 0.95 | 0.88 | 0.95 | 0.89 | |

## VI. DISCUSSION AND CONCLUSIONS

In various domains, initial UML Use Case sketches can still be drawn without a CASE tool by hand on paper, on a smartboard, or with freeform drawing tools. To further develop the initial sketch and share it conveniently with other stakeholders, the diagram sketch must be manually transferred to a UML standard supporting tool. This process, if performed manually, is inefficient, time-consuming, and may be prone to errors. Our proposed UML Use Case sketch digitization methodology can be used to facilitate requirements engineering by automating the digitization of UML Use Case diagram sketches.

The results of the experimental evaluation are promising: the recall for all element class recognition was 0.96, precision was 0.91, and the F1 score was 0.94. Additionally, an experiment aimed at evaluating the effectiveness of additional post-processing steps showed increased accuracy. The overall recall for all element digitization was 0.93, precision was 0.97, and F1 score was 0.95. While these results highlight the tool effectiveness in recognizing individual diagram elements, an analysis of the transformation process reveals several limitations that impact its performance, particularly in more complex diagrams. The primary source of errors comes from symbol overlap and inconsistent sketching styles. For instance, sketches with closely placed elements or symbols that overlap present challenges, leading to false positives in relation identification. This is problematic in complex diagrams with multiple associations, where incorrect element integration leads to a drop in accuracy. For example, when relation directions and the ability to map and connect different elements are considered, the recall for diagram element integration decreases to 0.84, precision to 0.95, and the F1 score to 0.89. This indicates that while the tool is effective in recognizing and digitizing individual elements with high precision and recall, there is still room for improvement in accurately integrating elements into a diagram structure, especially when non-standard layouts, such as curved, diagonal, overlapping lines, are present. Misalignment of key points in such cases leads to incorrect detection of start and end points for relationships, which in turn affects the accuracy of the final model. Moreover, the handwritten text recognition component struggles with legibility issues, especially when text overlaps with symbols, resulting in errors in labeling.

The tool performs best or can successfully achieve the best digitization results when the diagram sketches meet the following criteria:

- The sketch is on a neutral background, without shadows or elements unrelated to the Use Case diagram.
- Sketch does not have overlapping or non-straight relations.
- Sketch is specified according to the UML 2.5 specification.
- Sketch has legible element names.
- Sketch is of a high-resolution.

When these conditions are not met, the tool's performance declines. For instance, low-quality sketches or those containing shadows lead to an increase in false positives, and relations drawn with curved lines are often mapped incorrectly. This reveals a limitation in the current method of detecting foreground pixels for keypoint identification, particularly in low-resolution images where symbol boundaries are less distinct.

Regardless of the recognition and digitization accuracy, even if not complete, the sketch is digitized. The missing elements mean a lower precision, which is due to the low confidence being used, outlining the need for additional processing steps to discard redundant or incorrectly recognized elements. To enable higher recognition confidence and achieve higher precision, a more comprehensive dataset with as many variations of a specific element as possible, different diagram backgrounds, element sizes, rotations, and styles is needed.

Another limitation is the tool’s inability to handle non-standard Use Case diagrams effectively. Complex scenarios such as mixed layouts or non-adherence to UML specifications introduce variability that the current dataset does not cover. Expanding the dataset to include more complex scenarios would significantly improve the model’s ability to generalize. This would allow for better precision, particularly in situations where diagrams are drawn informally or with irregular layouts.

For now, postprocessing, or the main keypoint detection component, achieves 0.84 accuracy. The component checks the amount of foreground pixels at the corners. Although this primitive method achieves fairly high result, a more sophisticated method can be used to digitize diagrams of higher complexity, such as those with overlapping connections or connections composed of several curved lines. The current method of determining keypoint regions by foreground pixels may not work correctly if there is overlapping text or other element regions.

Although the tool demonstrates a high level of digitization accuracy under specific conditions, there is still room for improvement. Future work should focus on expanding the dataset, improving recognition models, and developing more sophisticated methods for keypoint and relation detection to further enhance the tool capabilities. Another area for exploration could involve the use of LLMs with visual recognition capabilities. While current research shows that LLMs require human intervention due to their limitations in reliably transforming diagrams into structured models, continued advancements in LLMs could help address the tool’s challenges in handling non-standard layouts or mixed-use diagrams. This approach, though still in its early stages, could improve flexibility in dealing with complex or informal sketches, provided these limitations are carefully managed.

## REFERENCES

[1] OBJECT MANAGEMENT GROUP, "UML 2.5.1 specification," Unified Modeling Language, v2.5.1. [Online]. Available: https://www.omg.org/spec/UML/2.5.1/PDF

[2] D. Rosenberg and M. Stephens, Eds., "Use Case Modeling," in *Use Case Driven Object Modeling with UML: Theory and Practice*, Berkeley, CA: Apress, 2007, pp. 49–82. doi: 10.1007/978-1-4302-0369-8_3.

[3] G. Booch, R. A. Maksimchuk, M. W. Engle, B. J. Young, J. Connallen, and K. A. Houston, "Object-oriented analysis and design with applications, third edition," *ACM SIGSOFT Softw. Eng. Notes*, vol. 33, no. 5, pp. 29–29, Aug. 2008, doi: 10.1145/1402521.1413138.

[4] I. Jacobson, G. Booch, and J. Rumbaugh, *The Unified Software Development Process*, First Edition. Reading, Mass: Addison-Wesley Professional, 1998.

[5] "Requirements Engineering: From System Goals to UML Models to Software Specifications | Wiley," Wiley.com. Accessed: Jun. 14, 2024. [Online]. Available: https://www.wiley.com/en-be/Requirements+Engineering%3A+From+System+Goals+to+UML+Models+to+Software+Specifications-p-9780470012703

[6] "Use Cases– Requirements in Context - Daryl Kulak, Eamonn Guiney - Google knygos." Accessed: Jun. 14, 2024. [Online]. Available: https://books.google.lt/books/about/Use_Cases.html?id=qOjxBo7gTLwC&redir_esc=y

[7] "Supporting use case based requirements engineering | Request PDF." Accessed: Jun. 14, 2024. [Online]. Available: https://www.researchgate.net/publication/222694189_Supporting_use_case_based_requirements_engineering

[8] W. Shen and S. Liu, "Formalization, Testing and Execution of a Use Case Diagram," Nov. 2003, pp. 68–85. doi: 10.1007/978-3-540-39893-6_6.

[9] Y. Shinkawa, "Model Checking for UML Use Cases," in *Software Engineering Research, Management and Applications*, R. Lee, Ed., Berlin, Heidelberg: Springer, 2008, pp. 233–246. doi: 10.1007/978-3-540-70561-1_17.

[10] "Early Validation and Verification of System Behaviour in Model-based Systems Engineering: A Systematic Literature Review | ACM Transactions on Software Engineering and Methodology." Accessed: Jun. 14, 2024. [Online]. Available: https://dl.acm.org/doi/10.1145/3631976

[11] M. Cherubini, G. Venolia, R. DeLine, and A. J. Ko, "Let's go to the whiteboard: how and why software developers use drawings," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, in CHI '07. New York, NY, USA: Association for Computing Machinery, balandžio 2007, pp. 557–566. doi: 10.1145/1240624.1240714.

[12] "About the XML Metadata Interchange Specification Version 2.5.1." Accessed: Jun. 14, 2024. [Online]. Available: https://www.omg.org/spec/XMI/

[13] J. F. Tavares, Y. M. G. Costa, and T. E. Colanzi, "Classification of UML Diagrams to Support Software Engineering Education," in *2021 36th IEEE/ACM International Conference on Automated Software Engineering Workshops (ASEW)*, Nov. 2021, pp. 102–107. doi: 10.1109/ASEW52652.2021.00030.

[14] B. Karasneh and M. R. V. Chaudron, "Extracting UML models from images," in *2013 5th International Conference on Computer Science and Information Technology*, Mar. 2013, pp. 169–178. doi: 10.1109/CSIT.2013.6588776.

[15] T. Ho-Quang, M. Chaudron, I. Samuelsson, J. Hjaltason, B. Karasneh, and M. H. Osman, "Automatic Classification of UML Class Diagrams from Images," Dec. 2014. doi: 10.1109/APSEC.2014.65.

[16] A. Conrardy and J. Cabot, "From Image to UML: First Results of Image Based UML Diagram Generation Using LLMs," Jul. 2024. doi: 10.48550/arXiv.2404.11376.

[17] J. Cámara, J. Troya, L. Burgueño, and A. Vallecillo, "On the assessment of generative AI in modeling tasks: an experience report with ChatGPT and UML," *Softw. Syst. Model.*, vol. 22, no. 3, pp. 781–793, Jun. 2023, doi: 10.1007/s10270-023-01105-5.

[18] G. Costagliola, V. Deufemia, and M. Risi, "A Multi-layer Parsing Strategy for On-line Recognition of Hand-drawn Diagrams," presented at the Proceedings - IEEE Symposium on Visual Languages and Human-Centric Computing, VL/HCC 2006, Jan. 2006, pp. 103–110. doi: 10.1109/VLHCC.2006.4.

[19] M. Bresler, D. Průša, and V. Hlavac, "Online recognition of sketched arrow-connected diagrams," *Int. J. Doc. Anal. Recognit. IJDAR*, vol. 19, Sep. 2016, doi: 10.1007/s10032-016-0269-z.

[20] V. Moreno, G. Génova, M. Alejandres, and A. Fraga, "Automatic Classification of Web Images as UML Static Diagrams Using Machine Learning Techniques," *Appl. Sci.*, vol. 10, no. 7, Art. no. 7, Jan. 2020, doi: 10.3390/app10072406.

[21] J. Fang, Z. Feng, and B. Cai, "DrawnNet: Offline Hand-Drawn Diagram Recognition Based on Keypoint Prediction of Aggregating Geometric Characteristics," *Entropy Basel Switz.*, vol. 24, no. 3, p. 425, Mar. 2022, doi: 10.3390/e24030425.

[22] B. Schäfer, H. van der Aa, H. Leopold, and H. Stuckenschmidt, "Sketch2Process: End-to-End BPMN Sketch Recognition Based on Neural Networks," *IEEE Trans. Softw. Eng.*, vol. 49, no. 4, pp. 2621–2641, Apr. 2023, doi: 10.1109/TSE.2022.3228308.

[23] M. Bresler, D. Průša, and V. Hlavac, "Modeling Flowchart Structure Recognition as a Max-Sum Problem," presented at the Proceedings of the International Conference on Document Analysis and Recognition, ICDAR, Aug. 2013, pp. 1215–1219. doi: 10.1109/ICDAR.2013.246.

[24] K. He, G. Gkioxari, P. Dollár, and R. Girshick, "Mask R-CNN," in *2017 IEEE International Conference on Computer Vision (ICCV)*, Oct. 2017, pp. 2980–2988. doi: 10.1109/ICCV.2017.322.

[25] V. Agrawal, J. Jagtap, and M. P. Kantipudi, "An Overview of Hand-Drawn Diagram Recognition Methods and Applications," *IEEE Access*, vol. 12, pp. 19739–19751, 2024, doi: 10.1109/ACCESS.2024.3357398.

[26] S. Shcherban, P. Liang, Z. Li, and C. Yang, "Multiclass Classification of UML Diagrams from Images Using Deep Learning," *Int. J. Softw. Eng. Knowl. Eng.*, vol.

31, no. 11n12, pp. 1683–1698, Dec. 2021, doi: 10.1142/S0218194021400179.
[27] M. Bresler, T. Phan, D. Průša, M. Nakagawa, and V. Hlavac, “Recognition System for On-Line Sketched Diagrams,” presented at the Proceedings of International Conference on Frontiers in Handwriting Recognition, ICFHR, Sep. 2014. doi: 10.1109/ICFHR.2014.100.
[28] O. Altun and O. Nooruldeen, “SKETRACK: Stroke-Based Recognition of Online Hand-Drawn Sketches of Arrow-Connected Diagrams and Digital Logic Circuit Diagrams,” *Sci. Program.*, vol. 2019, pp. 1–17, Nov. 2019, doi: 10.1155/2019/6501264.
[29] H. Liu *et al.*, “Flowmind2digital: The First End-to-End Flowmind Recognition and Conversion Approach,” Apr. 19, 2023, *Rochester, NY*: 4423352. doi: 10.2139/ssrn.4423352.
[30] Q. Chen, J. Grundy, and J. Hosking, “SUMLOW: early design-stage sketching of UML diagrams on an E-whiteboard,” *Softw. Pract. Exp.*, vol. 38, no. 9, pp. 961–994, 2008, doi: 10.1002/spe.856.
[31] L. Fu and L. B. Kara, “From engineering diagrams to engineering models: Visual recognition and applications,” *Comput.-Aided Des.*, vol. 43, no. 3, pp. 278–292, Mar. 2011, doi: 10.1016/j.cad.2010.12.011.
[32] B. Schäfer, M. Keuper, and H. Stuckenschmidt, “Arrow R-CNN for handwritten diagram recognition,” *Int. J. Doc. Anal. Recognit. IJDAR*, vol. 24, no. 1, pp. 3–17, Jun. 2021, doi: 10.1007/s10032-020-00361-1.
[33] “A Comprehensive Review of YOLO Architectures in Computer Vision: From YOLOv1 to YOLOv8 and YOLO-NAS.” Accessed: Oct. 05, 2024. [Online]. Available: https://www.mdpi.com/2504-4990/5/4/83
[34] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, “You Only Look Once: Unified, Real-Time Object Detection,” in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2016, pp. 779–788. doi: 10.1109/CVPR.2016.91.
[35] P. Jiang, D. Ergu, F. Liu, Y. Cai, and B. Ma, “A Review of Yolo Algorithm Developments,” *Procedia Comput. Sci.*, vol. 199, pp. 1066–1073, Jan. 2022, doi: 10.1016/j.procs.2022.01.135.
[36] A. Bochkovskiy, C.-Y. Wang, and H. Liao, *YOLOv4: Optimal Speed and Accuracy of Object Detection*. 2020. doi: 10.48550/arXiv.2004.10934.
[37] W. Liu *et al.*, “SSD: Single Shot MultiBox Detector,” in *Computer Vision – ECCV 2016*, B. Leibe, J. Matas, N. Sebe, and M. Welling, Eds., Cham: Springer International Publishing, 2016, pp. 21–37. doi: 10.1007/978-3-319-46448-0_2.
[38] J. Ni, R. Wang, and J. Tang, “ADSSD: Improved Single-Shot Detector with Attention Mechanism and Dilated Convolution,” *Appl. Sci.*, vol. 13, no. 6, Art. no. 6, Jan. 2023, doi: 10.3390/app13064038.
[39] Q. Yin, W. Yang, M. Ran, and S. Wang, “FD-SSD: An improved SSD object detection algorithm based on feature fusion and dilated convolution,” *Signal Process. Image Commun.*, vol. 98, p. 116402, Oct. 2021, doi: 10.1016/j.image.2021.116402.
[40] S. P. Yadav, M. Jindal, P. Rani, V. H. C. de Albuquerque, C. dos Santos Nascimento, and M. Kumar, “An improved deep learning-based optimal object detection system from images,” *Multimed. Tools Appl.*, vol. 83, no. 10, pp. 30045–30072, Mar. 2024, doi: 10.1007/s11042-023-16736-5.
[41] D. D. Aboyomi and C. Daniel, “A Comparative Analysis of Modern Object Detection Algorithms: YOLO vs. SSD vs. Faster R-CNN,” *ITEJ Inf. Technol. Eng. J.*, vol. 8, no. 2, Art. no. 2, Dec. 2023, doi: 10.24235/itej.v8i2.123.
[42] B. Bhavya Sree, V. Yashwanth Bharadwaj, and N. Neelima, “An Inter-Comparative Survey on State-of-the-Art Detectors—R-CNN, YOLO, and SSD,” in *Intelligent Manufacturing and Energy Sustainability*, A. N. R. Reddy, D. Marla, M. N. Favorskaya, and S. C. Satapathy, Eds., Singapore: Springer, 2021, pp. 475–483. doi: 10.1007/978-981-33-4443-3_46.
[43] J. Huang *et al.*, “Speed/Accuracy Trade-Offs for Modern Convolutional Object Detectors,” in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jul. 2017, pp. 3296–3297. doi: 10.1109/CVPR.2017.351.

**MANTAS RAŽINSKAS** earned both his BSc and MSc degrees in Computer Science from Kaunas University of Technology, graduating in 2019 and 2021 respectively. His research interests lie in UML modeling, Model-Driven Architecture (MDA), model transformations, code generation, and AI integration.

**BENAS MILIŪNAS** graduated with a BSc degree in Computer Science from Kaunas University of Technology in 2022. He furthered his studies and recently earned his MSc degree in 2024, also from the Faculty of Informatics at Kaunas University of Technology. His research interests lie in UML modeling, AI integration, software development, and distributed ledger technologies.

**MANTAS JURGELAITIS** earned both his BSc and MSc degrees in Computer Science from Kaunas University of Technology, graduating in 2016 and 2018, respectively. Most recently, in 2023, he received his PhD degree from the Faculty of Informatics at Kaunas University of Technology. His research interests lie in UML modeling, Model-Driven Architecture (MDA), model transformations, code generation, and software and blockchain-based systems engineering.

**LINA ČEPONIENĖ** received the B.Sc., M.Sc., and Ph.D. degrees from the Kaunas University of Technology, in 2000, 2002, and 2006, respectively. She is currently an Associate Professor and the Head of the Department of Information Systems, Faculty of Informatics, Kaunas University of Technology. Her research interests include UML modeling, MDA, model transformations, code generation, and blockchain-based systems engineering.

**LINA BISIKIRSKIENĖ** is an assistant professor at the Department of Information Systems, Faculty of Informatics, Kaunas University of Technology, and a Business Analyst Practice Lead at Cognizant Company. She received her BSc, MSc, and PhD in Informatics Engineering from the Kaunas University of Technology. She is an author and co-author of research papers in the domains of IT projects and business processes.