Xamã: Optical Character Recognition for Multi-domain Model Management

Weslley Torres, Mark G. J. van den Brand, and Alexander Serebrenik

Eindhoven University of Technology, Eindhoven, The Netherlands
{w.silva.torres,m.g.j.v.d.brand,a.serebrenik}@tue.nl

Abstract. The development of systems following model-driven engineering can include models from different domains. For example, to develop a mechatronic component one might need to combine expertise about mechanics, electronics, and software. Although these models belong to different domains, the changes in one model can affect other models causing inconsistencies in the entire system. Only few tools, however, support management of models from different domains. Indeed, these models are created using different modeling notations and it is not plausible to use a multitude of parsers geared towards each and every modeling notation. Therefore, to ensure maintenance of multi-domain systems, we need a uniform approach that would be independent from the peculiarities of the notation.
Notation-independence implies that such a uniform approach can only be based on elements commonly present in models of different domains, i.e., text, boxes, and lines. In this study we investigate the suitability of optical character recognition (OCR) as a basis for such a uniformed approach. We select graphical models from various domains that typically combine textual and graphical elements.
We start by analyzing the performance of Google Cloud Vision and Microsoft Cognitive Services, two off-the-shelf OCR services. Google Cloud Vision performed better than Microsoft Cognitive Services being able to detect text of 70% of model elements. Errors made by Google Cloud Vision are due to absence of support for text common in engineering formulas, e.g., Greek letters, equations, and subscripts. We identified the *multi-line text* error as one of the main issues of using OCR to recognize textual elements in models from different domains. This error happens when OCR misinterprets one textual element as two separate elements.
To address the *multi-line text* error we build Xamã on top of Google Cloud Vision. Xamã includes two approaches to identify whether the elements are positioned on a single line or multiple lines, and merge those identified as positioned on multiples lines. With and without shape detection Xamã correctly identified 956 and 905 elements, respectively, out of 1,171. Additionally, we compared the accuracy of Xamã and state-of-the-art tool img2UML, and we observe that Xamã outperformed img2UML in both precision and recall, being able to recognize 433 out of 614 textual elements as opposed to 171 by img2UML.

Keywords: Model Management · Systems Engineering · OCR.

1 Introduction

Model-driven engineering (MDE) has been used in diverse engineering fields such as software engineering [74], robotics [111], and automotive [103]. The promised benefits of using this approach include increased development speed, earlier system analysis, and more manageable complexity [109]. However, managing interrelated models of different domains is challenging [89]. Qamar et al. [105] recommend the explicit modeling of the relationships between these models as an approach to manage them.

A number of technologies have been proposed to model the relationships between models explicitly [85, 86, 105, 112]. However, automatic identification of these relationships remains an open problem. The main reason is the heterogeneity of modeling notations: we believe that it would not be feasible to develop a tool to parse all the existing notations. Moreover, even if such a tool was developed, it would have to be updated every time a modeling notation evolves or a new notation emerges as stated by Ruscio et al. [82]. Metamodels change over time [84], and the need to update the tools/models to support this change is known as the model co-evolution problem [100]. The literature reported that the time spent on this maintenance represents more than 25% of the total effort involved with creation of a Domain Specific Language (DSL) [100, 101]. Thus, we believe that to manage interrelated models of different domains one has to use a technology independent of the modeling language(s) used.

We observed that graphical models, independent of the engineering domain, typically combine textual and graphical elements such as boxes, lines, and arrows. Such models can be designed using different tools, that usually can export the model in a structured format such as XML, or as an image format such as PNG or JPEG [95]. From the data extraction perspective it would have been ideal if all models were available in a common structured format. However, this is often not the case as models might be only available as images [95, 104] as described in the following case:

- **Unavailability of the source code of the model.** Akdur et al. [73] identified that some models are discarded shortly after the engineer presents the model to a colleague. In order to increase the lifespan of the model, engineers take a picture of the model and store it as image [73, 75].
- **Models are stored as images.** Some companies prefer to store the models as images due to the impossibility, in the future, of opening the models in their original modeling tools [87]. This issue can happen when the source code of the model is available but the modeling tool that created the model cannot open it due to version compatibility. Maintaining the version compatibility between the source code of the model and the modeling tool can be an expensive, and resource-intensive process [87]. Similarly, engineers store 3D CAD models by exporting them to 2D drawings and saving the drawings as images [87]. This way of working is also common practice in Free/Open Source Software (FOSS) projects. Hebig et al. [90] investigated 3.295 GitHub projects and identified that more than 50% of UML files presented in those

projects are stored as images format (jpeg, png, gif, svg, bmp). Furthermore, a number of repositories reported in the literature store the models as image [91, 110]. These findings suggest that it is a common practice to store the models as images [91, 95, 96]
- **Engineers do not use formal modeling languages.** Akdur et al. [72] conducted a survey with 627 software engineers from 27 different countries on modeling and model-driven engineering practices in the embedded software industry. They identified that no formal modeling language was used to design some of the models by 65% of the participants. Examples of such models include those designed using analog media (papers, whiteboard), and also those models that were designed using computer tools but not necessary modeling tools, such as Microsoft PowerPoint. In this case, the engineers can either recreate the models using formal modeling tools, or they can easily export the model as an image or take a picture of the model.

Therefore, to ensure maintenance of multi-domain systems, we need a uniform approach that would be independent from the peculiarities of the notation. This also means that such a uniform approach can only be based on something which is present in all those models, i.e., text, boxes, and lines. We believe that the initial step to identify these relationships is through a uniform approach that extracts data presented in a number of models from different domains.

In the first part of this work, we investigate the suitability of optical character recognition (OCR) as part of this uniform approach independent from the peculiarities of the notation. OCR is a collection of techniques aiming at recognizing text from handwritten or printed document and exporting the result as a machine-encoded text. We start by evaluating two of the best[1] off-the-shelf OCR services, Google Cloud Vision[2] and Microsoft Cognitive Services[3], for extracting text from a collection of 43 models from different domains. We use precision, recall, and F-measure as metrics to evaluate the results. In our context, *precision* and *recall* are the fractions of OCR-extracted texts that are also manually extracted compared to either all OCR-extracted texts (*precision*), or compared to all manually extracted texts (*recall*). F-measure is the harmonic mean of precision and recall. Following are the research questions used to guide the first part of this work:

- **RQ1)** How accurate are off-the-shelf OCR services for extracting text from graphical models?
  - **Motivation:** While OCR techniques have been around since 1930s, they have not been applied in the context of text extraction from graphical models. Hence, it is crucial to evaluate accuracy of the state-of-the-art off-the-shelf OCR services with respect to these tasks.

[1] Top OCR by Accuracy, Price, and Capabilities. https://rapidapi.com/blog/top-5-ocr-apis/

[2] https://cloud.google.com/vision/

[3] https://azure.microsoft.com/en-us/services/cognitive-services/

  - **Answer:** We observe that Google Cloud Vision outperforms Microsoft Cognitive Services, being able to detect 70% of textual elements as opposed to 30% by Microsoft.
- **RQ2)** What are the common errors made by OCR services on models from different domains?
  - **Motivation:** Taking a closer look at the common errors made by Google Cloud Vision is a prerequisite to designing techniques that can improve the precision and recall of the OCR services when applied to text extraction from graphical models.
  - **Answer:** We organize the common errors made by Google Cloud Vision into four categories. The first category of errors is related to *non-alphanumerical characters* used in the models such as [, {, < or _. The second category is *mathematical notation* commonly used in equations such as subscripts and Greek letters. The following category of errors is related to *spacing* and relative positioning of the textual elements. Finally, the last group of errors is related to *single-character errors* such as characters being wrongly added, removed, or recognized. We observed that the main OCR challenges are related to recognizing text that contains equations, Greek letters, multi-line text, i.e., text fragments positioned on multiple lines, and subscripts.

Based on these findings in the second part of this work, we aim to improve the precision and recall of the OCR focusing on fixing the *multi-line text* error. We chose to correct this error for two main reasons: the first one is because OCR failed to detect textual elements that are positioned in multiple lines. The second reason is related to the long-term goal of our research which is the support for model management focusing on managing interrelated models of different domain. We believe that correcting errors related to mathematical formulas might not be as beneficial as correcting the *multi-line text* error. It is because even a small difference in one equation such as the presence of "-" instead of "+" can lead to a completely unrelated equation.

In order to address the *multi-line text* error, we developed XAMÃ (Figure 1) as an extra layer on top of Google Cloud Vision. This tool includes two approaches that identify whether the elements are positioned on a single line or multiple lines. As consequence, we merge those identified as positioned on multiples lines avoiding the *multi-line text* error. To achieve this goal, *i.e.*, identifying whether the text is positioned on multiples lines or not, we first investigated the similarities of textual elements presented in the models used in the first part of this work. Based on this analysis we defined a set of heuristics and applied them to a new collection of models to evaluate the accuracy of our approach. The second approach is a combination of a modified version of these heuristics with shape detection feature. The shape detection feature is a collection of image processing algorithms used to identify shapes such as boxes presented in the models.

Additionally, we evaluate the overall improvement of using XAMÃ and we compare the results to a state-of-the-art domain specific tool. We selected Img2UML

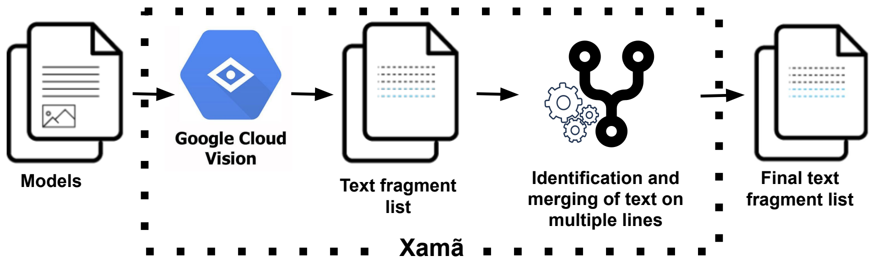

Fig. 1. XAMÃ is built on top of Google Cloud Vision. XAMÃ process the output provided by Google Cloud Vision to identify and merge the text fragments that are positioned on multiple lines. As output, XAMÃ produces the final text fragment list.

[95, 96] as the domain specific tool, and we compared the results by applying Img2UML to collection of 20 UML class diagrams. In this comparison, we do not take the identification of classes and relationships into account because the focus of this paper is to evaluate the use OCR in extracting text from models. Following are the research questions used to guide the second part of this work:

- **RQ3)** How accurate are the heuristics of XAMÃ in identifying multi-line problems?
  - Motivation: Evaluating the accuracy of the heuristics of XAMÃ is important to guarantee that they are not cause worse precision and recall values.
  - Answer: We applied our heuristics in a collection of 51 models (10 Matlab Simulink models, 20 UML diagrams, and 21 models from scientific papers). Our heuristics without shape detection correctly identified 905 out of 1171 elements, presenting a precision of 75%, recall of 77%, and f-measure of 76%. With shape detection the number of correctly identified elements increases to 956, improving the precision, recall and f-measure to to 84%, 82%, and 83% respectively. Evaluating the results by the model domain, we observed that XAMÃ, with the shape detection feature, presents statistically higher precision and recall on Matlab Simulink and the models in the scientific papers than without this feature. The results are inconclusive for UML diagrams.
- **RQ4)** What is the overall impact, in terms of precision and recall, of using our approaches?
  - Motivation: Due to the existence of other kind of errors, correcting the *multi-line text* error does not guarantee the improvement of overall accuracy.
  - Answer: XAMÃ, without shape detection, presents better results in 22 (out of 51) models in terms of precision, and in 16 models in terms of recall. With shape detection, it presents better result in terms of precision in 29 models, and recall in 22 models. Evaluating these results by the domain, we observed that XAMÃ presents statistically better precision

on Matlab Simulink and the models in the scientific papers. Regarding the recall, XAMÃ presents statistically better results on these two groups of models when using the shape detection feature. For UML diagrams the results are inconclusive.

- **RQ5)** How accurate is XAMÃ compared to a state-of-the-art domain specific tool?
  - Motivation: Since it is expected that a domain specific tool would outperform a generic tool we would like to know how good is XAMÃ compared to a domain specific tool regarding the text recognition.
  - Answer: We observed that XAMÃ correctly recognized 433/431 (without/with shape detection) out of 614 elements on class diagrams, while Img2UML correctly recognized 171 elements.

This paper is an extension of our previous work [113]. While the conference paper focused on the evaluation of the existing OCR techniques (**RQ1** and **RQ2**), in this extension, we propose two approaches to improve the OCR precision and recall focusing on correcting the error caused by the misinterpretation of one textual element as two separate elements (**RQ3**, **RQ4** and **RQ5**).

To encourage replication of our work the data we have collected and the source code we have used to perform the analysis have been made available on: bit.ly/DataOCRExtension

The remainder of this paper is organized as follows. Section 2 presents our previous study aimed at investigating the suitability of OCR in extracting text from models from different domains. In Section 3, we present the second study in which we address the *multi-line text* error. Section 4 presents the threats to validity, and the actions we took in order to mitigate the threats. In Section 5, we present a summary of our studies, discussing the results and indicating possible future work directions. Section 6 presents the related work. Finally, the conclusion is presented in Section 7.

2 Suitability of Optical Character Recognition for Text Extraction from Graphical Models

In this section, we present our previous study [113] aimed at investigating the suitability of optical character recognition (OCR) as a uniformed approach to extract data from models from different domains. Specifically, we investigate the accuracy of off-the-shelf OCR services for extracting text from graphical models (RQ1), and the common errors made by OCR (RQ2). In Section 3, we build upon this study and propose two approaches to correct the most common error produced by OCR.

2.1 Methodology

To answer RQ1 we apply Google Cloud Vision and Microsoft Cognitive Services to a collection of 43 models from different domains. To answer RQ2 we focus on the OCR service that has been shown to perform better on RQ1 and inspect the errors made by the service.

Models Selection For reproducibility reasons, we arbitrarily select models from two UML open repositories [1, 69], three control system engineering papers [71, 93, 114], and the example catalog of MatLab Simulink[4]. In total we analyzed 43 models as presented in Table 1. We only require the models to be graphical models, i.e., they must contain a mix of textual and graphical elements. Diagrams are graphical representations of parts of a model [70]. Therefore, in the context of this study, we use the term "model" to represent diagrams as well.

We select MatLab Simulink models because of its high adoption by the industry. These models are available on the official website as example catalog and they are used to describe control systems from different domains including automatic climate control, robot arm control, and fault-tolerant fuel control. We also include models from three scientific papers on control system engineering. The models from these papers are an intelligent control architecture of a small-scale unmanned helicopter, an actuator control system, and a x-ray machine.

Among the UML models we focus on Class Diagram, Sequence Diagram, and Use Case Diagrams. These models are stored in two repositories: Git UML [1] and Models-db [69]. The former automatically generates diagrams from source code stored in git repositories. Models-db is automatically populated by crawlers identifying models from public GitHub repositories.

Source	Models	#Models
Ai et al. [71]	Figures 1, 4, 5	5
Kaliappan et al. [93]	Figures 1-3, 5, 7	5
Tovar-Arriaga et al. [114]	Figures 1, 5–8, 10, 15	7
UML	[22, 32, 43, 45–51, 53–55, 60, 62–64]	17
MatLab Simulink	[3, 13–20]	9
Total		43

Table 1. List of models used to answer RQ1.

Text Extraction In order not to bias the evaluation towards a specific engineering domain, we opt for general-purpose OCR techniques. Several OCR serves are available off the shelf, including Google Cloud Vision, Microsoft Cognitive Services, and Amazon AWS Rekognition[5]. For this work, we select the Google Cloud Vision and Microsoft Cognitive Services: these services have been shown to be effective in recognizing text from the photos of the pages of the Bible [107], and to outperform Amazon AWS on images of business names or movie names [2].

[4] https://www.mathworks.com/products/simulink.html
[5] https://aws.amazon.com/rekognition/

Measures for Accuracy The validation consists of manually identifying the text from graphical models, and comparing the text extracted by OCR to the manually identified text. When deciding whether the OCR-extracted text matches the one manually extracted we do not distinguish between the letter case, i.e., *Velocity* is seen as the same as *veLoCitY* because these words still have the same meaning. We do distinguish between differently chunked texts, i.e., given the manually identified text *Velocity control* an OCR service extraction of *Velocity* and *Control* as two separate texts will be seen as wrong.

As common in information retrieval tasks we report precision, recall, and F-measure, i.e., the harmonic mean of precision and recall. In our context *precision* is the fraction of OCR-extracted texts that are also manually extracted compared to all OCR-extracted texts, and *recall* is the fraction of OCR-extracted texts that are also manually extracted compared to all manually extracted texts.

2.2 Results

RQ1: How accurate are off-the-shelf OCR services for extracting text from graphical models? In overall, Google Cloud Vision correctly detected 854 out of 1,232 elements, while Microsoft Cognitive Services correctly detected 388 elements. This observation concurs with previous evaluations of these OCR services. Indeed, on the photos of the pages of the Bible Reis et al. [107] observed that Google Cloud Vision had a relative effectiveness of 86.5% as opposed to 77.4% of Microsoft Cognitive Services. On images of business names or movie names [2] Google Cloud Vision achieved 80% of both precision and recall as opposed to 65% of precision and 44% of recall of Microsoft Cognitive Services.

Hence, we hypothesize that also on our dataset Google Cloud Vision will outperform Microsoft Cognitive Services in terms of both precision and recall. Formally, we state the following hypotheses:

– H_0^p: The median difference between the *precision* for Google Cloud Vision and Microsoft Cognitive Services is zero.
– H_a^p: The median difference between the *precision* for Google Cloud Vision and Microsoft Cognitive Services is greater than zero.
– H_0^r: The median difference between the *recall* for Google Cloud Vision and Microsoft Cognitive Services is zero.
– H_a^r: The median difference between the *recall* for Google Cloud Vision and Microsoft Cognitive Services is greater than zero.

To test these hypotheses we perform two paired Wilcoxon signed-rank tests, one for precision and another one for recall. The p-values obtained for precision and recall are 1.9×10^{-7} and 2.8×10^{-9}, respectively. Hence, we can reject H_0^p and H_0^r and state that Google Cloud Vision outperforms Microsoft Cognitive Services.

To illustrate this argument consider Figure 2. It summarizes precision (y-axis) and recall (x-axis) organized by the type of models. Indeed, we can observe that while precision and recall obtained by Google Cloud Vision *mostly* exceed 0.5, precision and recall obtained by Microsoft Cognitive Services are *mostly* below

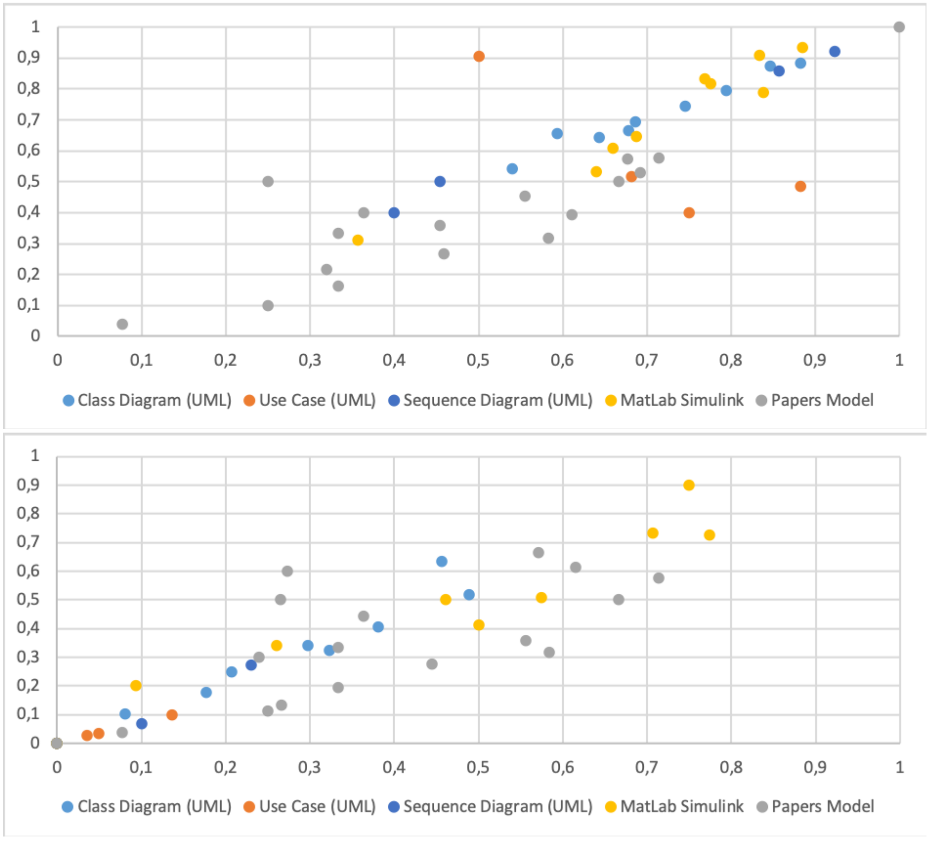

Fig. 2. Precision (y-axis) and recall (x-axis) obtained by Google Cloud Vision (top) and Microsoft Cognitive Services (bottom).

0.5. Moreover, the data for both Google Cloud Vision and Microsoft Cognitive Services suggests a linear relation between precision and recall: indeed, while the *number* of textual elements extracted by OCR tools is often close to the number of manually identified textual elements, the textual elements themselves are imprecise.

Finally, while Google Cloud Vision extracted *some* textual information from all models, Microsoft Cognitive Services failed on two models: Matlab Simulink model [15] and Figure 4.b from the paper by Ai et al. [71].

Performance on Models of Different Domains While the previous discussion indicates that overall Google Cloud Vision outperforms Microsoft Cognitive Services, *a priori* this does not imply that this should also be the case for models of different domains. This is why we formulate the corresponding hypotheses separately for UML diagrams, Matlab Simulink models, and models from scien-

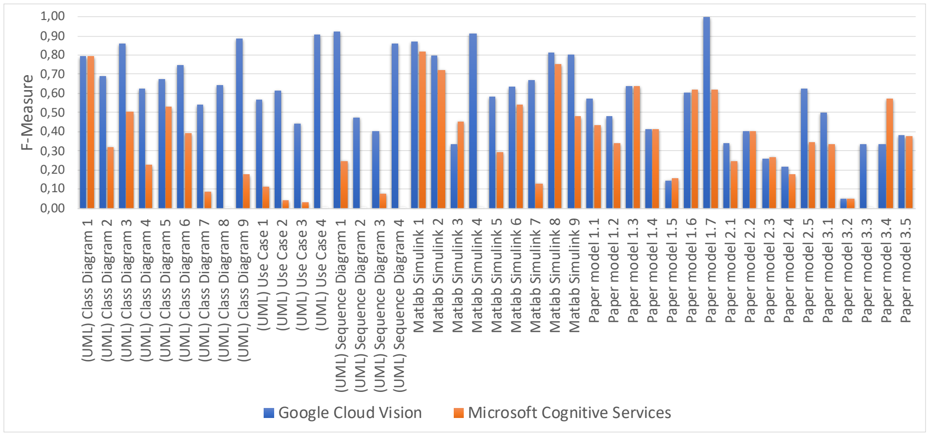

Fig. 3. F-Measure for all analyzed models. The mapping of paper model ID and the models from the selected scientific papers is presented in the Appendix Table 11

tific papers. We test these hypotheses using paired Wilcoxon signed-rank tests, one for precision and another one for recall. However, since we perform multiple comparisons, we need to adjust the p-values to control for the false discovery rate. We use the method proposed by Benjamini and Hochberg [77].

The adjusted p-values are below the commonly used threshold of 0.05 for five of the six comparisons (three types of models × precision or recall). We conclude that Google Cloud Vision outperforms Microsoft Cognitive Services: for models of all domains in terms of recall; for UML diagrams and Matlab Simulink models in terms of precision as presented in Table 2.

Models	Precision	Recall
UML	Google Cloud Vision	Google Cloud Vision
Matlab Simulink	Google Cloud Vision	Google Cloud Vision
Scientific papers	-	Google Cloud Vision

Table 2. OCR service that presents statistically better results organized by the domain. The "-" means inconclusive result.

Performance on Individual Models Table 2. Figure 3 shows that the F-measure for Google Cloud Vision is higher than for Microsoft Cognitive Services on 33 models, as opposed to five models where Microsoft Cognitive Services scores higher. For the remaining five models the F-measures are equal.

Inspecting Figure 3 we also notice that for six models Microsoft Cognitive Services have precision equal to zero, i.e., either no textual elements have been extracted (Matlab Simulink 4 and Paper model 3.3) or all textual elements extracted are wrong (UML Class Diagram 8, UML Use Case 4, UML Sequece Diagram 2 and 4). Unfortunately, we cannot precisely state the reasons why Microsoft Cognitive Services failed in process these models. Possible reasons could be related to the quality of the images, and size of font. However, these are unlikely to be the reasons for this fail, since all used images are in good quality and Google Cloud Vision managed to process the same images. In this study we did not look further in investing the reasons for the bad performance of Microsoft Cognitive Services.

Take away message

Google Cloud Vision was capable of detecting 70% of all elements, consistently outperforming Microsoft Cognitive Services.

RQ2: What are the common errors made by OCR services on models from different domains? Based on our answer to RQ1, one would prefer Google Cloud Vision as the OCR service to be integrated in a multi-domain model management solution. In this section we take a closer look at the errors made by Google Cloud Vision: addressing these errors is necessary in order to make OCR suited for multi-domain model management.

Table 3 summarizes the results of manual analysis of the errors made by Google Cloud Vision:

– The first category of errors is related to *non-alphanumerical characters* used in the models such as [, {, <, or _. These characters are sometimes confused with each other or missed by the OCR, e.g., the name of the element is *'file_version'* and OCR detects *'file version'*, without the underscore.
– Engineering models can involve *mathematical formulas* such as equations, including subscripts and Greek letters.
– The next group of errors is related to *spacing* and relative positioning of the textual elements. For example, due to space limitations text can be positioned on multiple lines, making OCR to misinterpret as one textual element but as two separate elements, we call this error as *Multi-line Text*. When this misinterpretation happens in a textual element positioned in one single line, we call this error as *Split Element*. The difference between *Multi-line Text* and *Split Element* is that the latter occurs on textual element written in one single line but have an empty space between the words, causing this misinterpretation. The opposite of the error *Split Element* is *Mix of Elements*. *Mix of Elements* occurs when OCR mixes the name of different elements due to their proximity.
– Finally, the last group of errors is related to single-character errors such as characters being wrongly added, removed, or recognized. An example of such error is *Character Confusion*. This error occurs when OCR is not capable of identifying the letter due to the similarity to other letters. For instance,

Problem	Total	UML CD	UML UC	UML SD	MatLab	Paper
Non-alphanumeric characters						
Brackets	1	0	0	0	1	0
Curly Brackets	2	0	0	2	0	0
Greater/Less Symbol	1	0	1	0	0	0
Parentheses	5	3	0	1	1	0
Slash	1	0	0	1	0	0
Underscore	8	7	0	0	1	0
Total	*18*	*10*	*1*	*4*	*3*	*0*
Mathematical formulas						
Equation	2	0	0	0	2	0
Subscript	2	0	0	0	0	2
Greek Letter	2	0	0	0	0	2
Total	*6*	*0*	*0*	*0*	*2*	*4*
Spacing						
Empty Space between Letters	15	6	4	3	1	1
Mix of Elements	8	4	0	0	3	1
Multi-line Text	28	4	3	0	7	14
Split Element	1	1	0	0	0	0
Total	*51*	*15*	*7*	*3*	*11*	*15*
Character confusion						
Character Confusion	8	1	2	0	3	2
Extra Char	11	4	2	0	2	3
Missing Char	14	5	1	0	3	5
Wrong Char	13	3	0	2	5	3
Total	*46*	*13*	*5*	*2*	*13*	*13*

Table 3. Number of models affected by the identified problems. CD - Class Diagram, UC - User Case, SD - Sequence Diagram

the name of the element is *'DeleteNodeById()'*. However, OCR interprets the capital letter 'i' as the lowercase 'l', returning *'DeleteNodeByld()'*. The difference between *Character Confusion* and *Wrong Char* is that the former occurs between similar characters, e.g., the letter 'o' and the number '0'. And *Wrong Char* occurs between any character.

Table 3 shows that errors present in the largest number of models are *Multi-line Text*, *Empty Space between Letters*, *Missing Char*, and *Wrong Char*. However, the number of models affected by the errors should be compared to the number of models that *can* be affected by those errors: while wrong characters might appear in any model, errors related to underscores can only be present if the models contain underscores.

Hence, Table 4 summarizes the number of models that can be affected (candidate models) and the models that are affected by errors. Similarly, it includes the number of elements that can be affected (candidate elements) and are affected by the errors.

Problem	#Affected Models	#Candidate Models	#Affected Elements	#Candidate Elements
Non-alphanumeric characters				
Brackets	1	1	1	5
Curly Brackets	2	2	8	9
Greater/Less Symbol	1	12	4	60
Parentheses	5	5	11	137
Slash	1	11	1	39
Underscore	8	8	59	156
Mathematical formulas				
Equations	2	2	2	2
Subscript	2	2	15	15
Greek letters	2	2	2	2
Spacing				
Multi-line Text	27	27	130	130
Split Element	1	39	2	334

Table 4. Candidate Elements are the elements that contain characters that can cause a problem. Candidate Models are the models that have the candidate elements.

Inspecting Table 4 we observe that the *Curly Brackets*, *Equations*, *Greek letters*, *Multi-line String*, *Parentheses*, *Subscript*, and *Underscore* occur in every single model that has the corresponding elements.

Even though *Parentheses*, and *Underscore* problems arise in 100% of the candidate models, Google Cloud Vision correctly identified 92% of textual elements that have parentheses and 60% of the textual elements that have underscores and this in sharp contrast with *Equations*, *Greek Letters*, *Multi-line String*, and *Subscript* that could not be recognized by Google Cloud Vision.

Take away message

The main OCR challenges are text that contains *Equations*, *Greek Letters*, *Multi-line String*, and *Subscript* due to the lower precision on correctly identify these elements.

3 Addressing an OCR Limitation - The Multi-line Error

The long-term goal of our research is to support the model management focusing on managing interrelated models of different domains. It is known that the *explicit* modeling of the relationships between models from different domains can support this goal. Indeed, explicit modeling of the relationships facilitates the identification of the affected models due to a change. While explicit modeling of the relationships can be done using a number of approaches, little is known in how to do this automatically. For this goal, we first need an approach to extract information that is presented in various kind of models. In Section 2, we investigated the use of OCR as a basis for such approach, and the common errors produced by OCR.

In this section we improve the precision of OCR by correcting the errors caused by the misinterpretation of one textual element as two separate elements. We call this error as the *multi-line text* error. We decided to correct this kind of error first because OCR failed in recognize these elements in 100% of the cases. Although the identification of mathematical formulas also represents the main OCR challenges, we believe that fixing errors related to mathematical formulas might not be as beneficial to the long-term goal of our research as correcting the *multi-line text* error. The reason is because even a small difference in one equation such as the presence of "-" instead of "+" can lead to a completely unrelated equation making it difficult to identify relationships between models.

3.1 Methodology

We observed that OCR services fail in detecting textual elements that are positioned on multiple lines. In order to address this error, we developed XAMÃ. This tool includes two approaches: the first approach is the use of a collection of heuristics that take into consideration parameters such as the text alignment, the distance between the text fragments, and the size of the words and letters. The second approach is the combination of these heuristics with image processing. When defining these approaches we aimed at solutions that would be independent of OCR engines and that would not introduce considerable overhead in terms of running time or memory consumption.

Identifying Similarities The main goal of the second part of this study is to propose a generic solution to the *multi-line text* error. We need a generic solution because of the heterogeneity of models. The first step to fix the *multi-line text* error is to identify the text that is positioned on multiple lines. For the sake of generalisability of the approach to be designed we focus on common characteristics of the text positioned on multiple lines. To identify these characteristics we analyzed the collection of models used to answer RQ1 and RQ2, and checked the characteristics of the multi-line text. Based on the characteristics, we defined a collection of heuristics capable of detecting text fragments positioned on multiple lines.

A possible approach to resolving multi-line errors might consist in merging all text fragments that are located on top of and close to each other. However, this approach would be overly eager and would merge unrelated text fragments, e.g., attributes and operations in a UML class diagram such as the one on Figure 4.

We propose two approaches to correct the *multi-line text* error. The first approach applies a set of heuristics used to classify whether the text should be merged taking into consideration the coordinates of the text fragments. Following are the description of these heuristics we call *regular heuristics (RH)*:

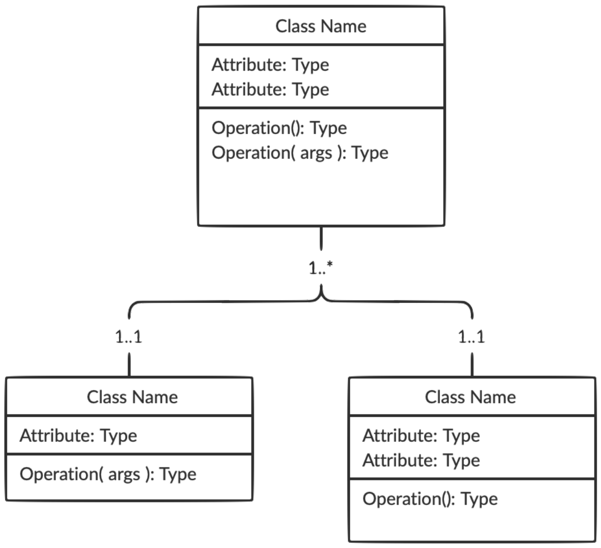

Fig. 4. Example of a UML Class Diagram.

- **Alignment**. We observed that the text fragments that are positioned on multiple lines have *centralized* as the chosen alignment format in most of the models (25 out of 28). Therefore, we opted to take this alignment format into consideration when evaluating whether the text fragment should be merged or not.
- **Text Distance**. Text fragments positioned below each other do not necessary represent the same element. To define whether the text fragments should be merged, we take into consideration the font size (height) and the distance (horizontal and vertical) between the text fragments. The threshold for the vertical distance between the text fragments is half of the font size. The horizontal distance is the distance between the centers of the both texts fragments. In this study, we consider that the threshold for the horizontal distance between the centers of the text fragments cannot be greater than 3 times the length of the letters[6]. These thresholds have been chosen in order to obtain the optimal performance on the data set described in Section 2. For example, we observed that increasing the threshold of the horizontal distance to 4 times the length of the letters leads to an increase of false positives. Figure 5 illustrates an example of how we calculate the text distance.

The second approach consists of a slightly modified version of the previous heuristics, and image processing algorithms to detect shapes presented in the models. For the image processing we use OpenCV [21], an open source com-

[6] The length of the letters is the length of the word divided by the number of characters in the word.

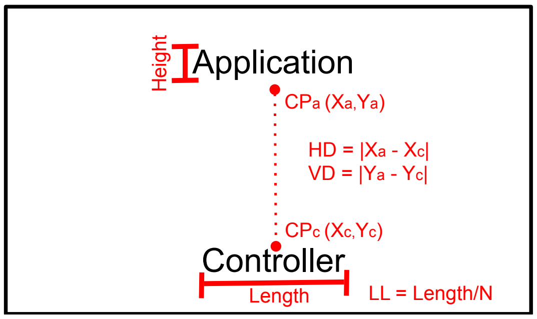

Fig. 5. Example of how we calculate the variables used in our heuristics of two text fragments that represent the element "Application Controller". The "CP" means central point of the word. The "HD" and "VD" mean horizontal and vertical distances respectively. The "LL" means length of the letters. The "N" means the number of letters in a word.

puter vision and machine learning software library. This library has more than 2500 optimized algorithms and is used by companies such as Google, Yahoo, Microsoft, Intel, IBM, Sony, Honda, and Toyota, with estimated number of downloads exceeding 18 million [21].

In order to improve the quality of the shape detection, we first convert the images to black and white, and then we blur these images using the Gaussian kernel filter [88] reducing the high-frequency noise from images. To identify the shapes presented in the images, we use the algorithm *findContours* with simple approximation as the contour approximation method. We opt for simple approximation to reduce memory consumption[7]. Once we have a list of the identified shapes, we use the *pointPolygonTest* algorithm to perform a point-in-contour test to determine whether the text fragments (provided by Google Cloud Vision) are inside the identified shapes.

Next we apply a series of heuristics. Instead of applying *RH* as defined above, we slightly modify them. Indeed, grouping the text fragments into the shapes they belong to, allows us to apply the heuristics in a less strict manner. While in the *RH* the threshold for the vertical distance is half of the font size, for the modified heuristics, we call *shape detection heuristics (SDH)*, the new threshold becomes the font size. The change of this value was derived by trying different values to obtain optimal results to improve the accuracy. This is the only difference between the heuristics used in both approaches.

The merging process is done on two text fragments at a time. For example, consider the following model in Figure 6. There are three elements in this model: *Navigation System*, *Application Controller*, and *Engine Speed Selector*. Because of the text is positioned on multiples lines, the OCR service recognizes *Application Controller* as two elements "Application" and "Controller"; and *Engine Speed Selector* as three elements "Engine", "Speed" and "Selector". In this ex-

[7] https://docs.opencv.org/master/d4/d73/tutorial_py_contours_begin.html

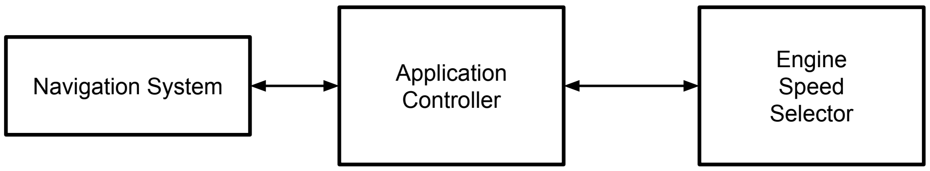

Fig. 6. An example of a model in which the *multi-line text* error occurs on *Application Controller* and *Engine Speed Selector* elements.

ample, XAMÃ starts checking whether "Navigation System" should merge with "Application", "Controller", "Engine", "Speed", and "Selector". In case of a merging, the merged text fragments become one, and we repeat the operation until all text fragments are checked. The merging is only considered correct if the order of the text fragments is correct. Having the element *Application Controller* as example, the merging is correct if the final result is "Application Controller" and wrong if it is "Controller Application".

We selected Google Cloud Vision as the OCR technology because of its high precision and recall as shown in the previous section. It is worth mentioning that XAMÃ works with any OCR service, as long as the service also provides the coordinates of the text fragment presented in the image. Our approach works as follows: we submit a collection of models to Google Cloud Vision that returns the list of text fragments and their coordinates in respect to the their models. XAMÃ processes the list of text fragments following a set of heuristics (with or without shape detection feature to identify and merge the text that is positioned on multiple lines, and exports a new list of text fragments. Every text fragment is one element of the model. Finally, we compare the final version of the text fragments with the ones manually extracted. These steps are summarized in Figure 7.

Limitations When we submit an image to be processed by Google Cloud Vision, we obtain as response the recognized text and the coordinates (x, y) in pixels of the text in the image. XAMÃ uses these coordinates to verify whether the text should be merged. However, in our preliminary experiments we observed that these coordinates are not always reliable. For example, considering the coordinates (x, y) of two textual elements that are aligned to the left. We would expect that the x values would be the same for both of textual elements. However, it is not always true. Thus, we have to take into account a tolerance range.

Another limitation of Google Cloud Vision is that one cannot specify the Google Cloud Vision version they want to use. This can be problematic because the results Google Cloud Vision, including coordinates computed on the same input model, changed with time. We conjecture that it happened due to an update of the service. However, this limitation would occur when using any

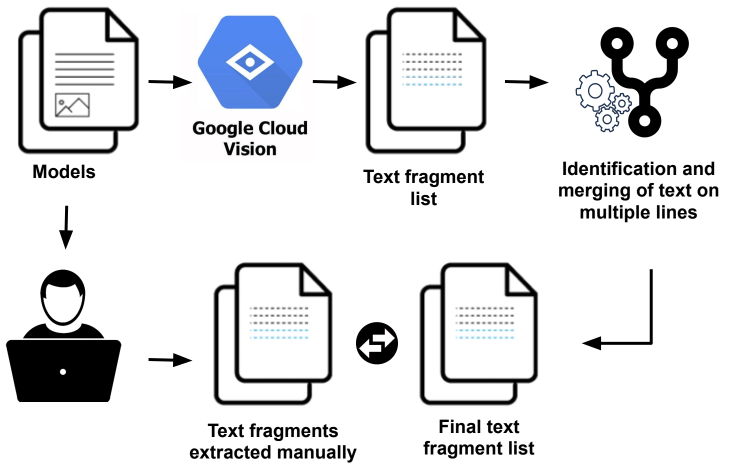
Models
Google Cloud Vision
Text fragment list
Identification and merging of text on multiple lines
Text fragments extracted manually
Final text fragment list

Fig. 7. Google Cloud Vision extracts a list of text fragments from a set of models. XAMÃ receives this list as input, processes it to identify and merge the text fragments that are positioned on multiple lines. As output, XAMÃ produces a final text fragment list to be compared with the text fragments extracted manually.

other external API/service where it is not possible to specify the version to be used.

Models Selection The methodology used to select the models to answer RQ3, RQ4, and RQ5 is similar to the one used to answer the first two research questions. The differences are the number of models. For this study, we selected 51 models instead of 43, and we used a different repository for UML sequence diagrams. In the previous models selection, we found five sequence diagrams in the Models-DB repository. However, for this study we could not find additional five sequence diagrams, we found only one in the Models-DB repository. The other four sequence diagrams we arbitrarily selected from the Lindholmen Dataset [90].

The selected models are organized as follows: 21 models from the scientific papers, 10 models from the example catalog of MatLab Simulink, and 20 UML diagrams (10 class diagrams, five user case diagrams, and five sequence diagrams). The majority (70%) of the models of this dataset presents text fragments positioned on multiple lines. It is worth mentioning that the presence of multi-line text was not a selection criterium for the models. Table 5 presents the models used in this study and the number of models per type that have text on multiple lines.

We also decided to include models that do not present text fragments positioned on multiple lines. The reasoning behind this decision is because there is a possibility of merging text fragments that should not be merged, causing false positives.

Source	Models	#Models presenting text positioned on multiple lines
Blasera et al. [79]	Figures 1, 3	2
Dounis et al. [83]	Figures 1-3, 5	3
Zhou et al. [117]	Figures 1b - 4, 6, 11, 13, 15, 19, 20	10
Zhu et al. [118]	Figures 1 - 3a, 5, 6	5
Class Diagram (UML)	[23–31, 33]	3
Sequence Diagram (UML)	[52, 56–59]	2
Use Case (UML)	[61, 65–68]	2
MatLab Simulink	[4–12]	9
Total	51	36

Table 5. List of models used in this study

3.2 Results

RQ3 - How accurate are our heuristics in identifying multi-line problems? XAMÃ was designed to identify whether the text is positioned on multiple lines and merge the fragments that should be merged correcting the *multi-line text* error. It is important to stress that in the following analysis we ignore other errors such as *character confusion* and *wrong char*. Therefore, to evaluate the precision and recall we consider as true positive those text fragments that XAMÃ correctly identified as positioned on multiple lines, and also those that XAMÃ identified as positioned on one single line. The reason for this is that it is possible that XAMÃ misinterprets two different text fragments as if they were only one but positioned on multiple lines, causing false positive.

Without Shape Detection We first evaluate our approach without the shape detection feature. As results, XAMÃ correctly identified 905 out of 1171 elements, with the overall precision of 75%, recall of 77%, and f-measure of 76%. Figure 8 (top) presents the distribution of these values. We observe that the majority of models that present precision lower than 70% are models from scientific papers. The reasons for low precision are: most of the text fragments are composed by long sentences, for instance: "Store the best value for the regularization parameter"; the presence of mathematical equations or subscripts; and the text distance between the text fragments is higher than the threshold defined in the *RH*. Most of the models ≈66% are located on the upper right side of the plot, meaning that they present precision and recall higher than 70%.

We also used f-measure as a metric to evaluate the accuracy of XAMÃ. The f-measure values are presented in Figure 9. We observe that ≈66% of the analyzed

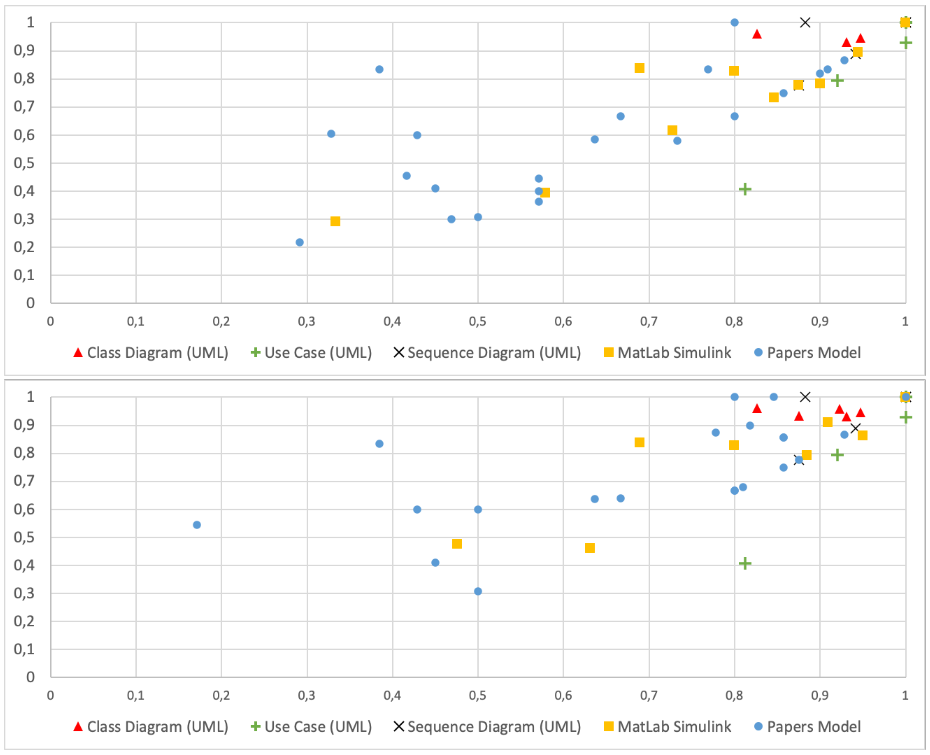

Fig. 8. Precision (y-axis) and recall (x-axis) obtained by XAMÃ without shape detection (top), with shape detection (bottom). In this analysis we focus on *Multi-line text* and we ignore other error types. There are 12 models (7 Class Diagrams, 2 Sequence and Use Case Diagrams, and 1 Matlab Simulink) with perfect precision and recall without shape detection, and 13 models (5 Class Diagrams, 2 Sequence Diagrams and Use cases, 3 Matlab Simulink, and 1 model from a scientific paper) with shape detection.

models have f-measure values greater than 70%. UML diagrams are the models that present higher values, and models from scientific papers are the models that present lower values.

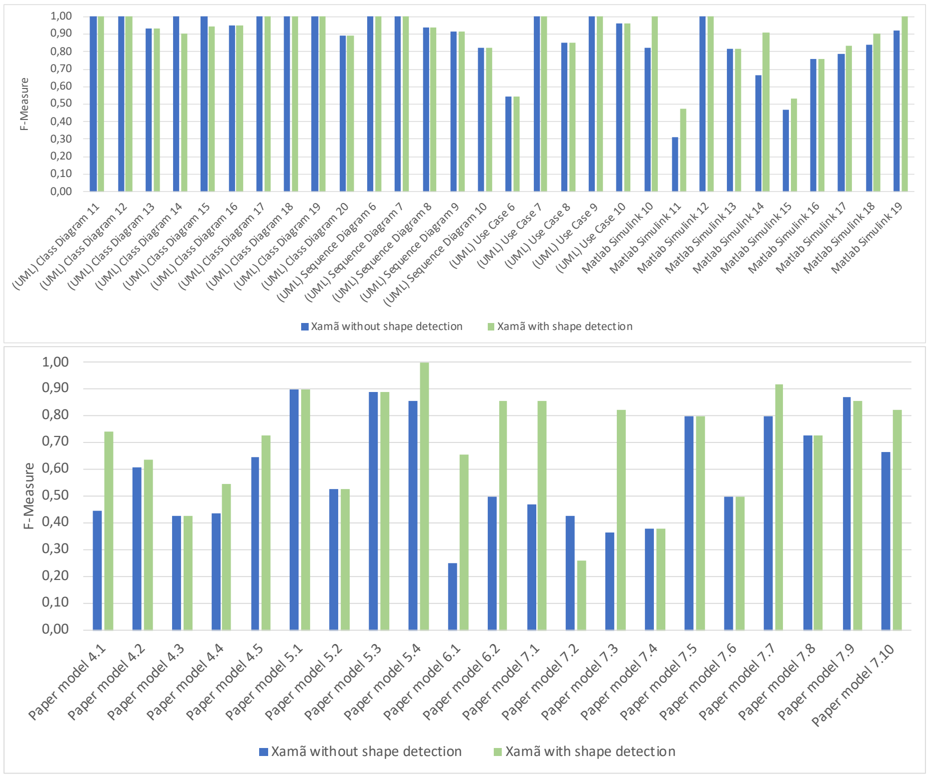

Fig. 9. F-Measure values focusing on the identification and merging of text positioned on multiple lines. The mapping of paper model ID and the models from the selected scientific papers is presented in the Appendix Table 11.

With Shape Detection When applying the shape detection feature, XAMÃ correctly identified 956 elements, improving the overall precision, recall and f-measure to 84%, 82%, and 83% respectively. Figure 8 (bottom) presents the distribution of these values. We observe that five models present precision and recall below 50%. Without the shape detection the amount of models presenting values below 50% are: 11 models for precision and eight for recall.

With or Without? Performance of XAMÃ with and without shape detection is presented in Table 6. By using shape detection we improved the precision in

	Precision		Recall		F-Measure	
Model	Heuristics	SD	Heuristics	SD	Heuristics	SD
Class Diagram 14	1.00	0.93	1.00	0.88	1.00	0.90
Class Diagram 15	1.00	0.96	1.00	0.92	1.00	0.94
Matlab Simulink 10	0.78	1.00	0.88	1.00	0.82	1.00
Matlab Simulink 11	0.29	0.48	0.33	0.48	0.31	0.48
Matlab Simulink 14	0.62	0.91	0.73	0.91	0.67	0.91
Matlab Simulink 15	0.39	0.46	0.58	0.63	0.47	0.53
Matlab Simulink 17	0.73	0.79	0.85	0.88	0.79	0.84
Matlab Simulink 18	0.78	0.86	0.90	0.95	0.84	0.90
Matlab Simulink 19	0.89	1.00	0.94	1.00	0.92	1.00
Paper [118] Figure 1	0.36	0.68	0.57	0.81	0.44	0.74
Paper [118] Figure 2	0.58	0.64	0.64	0.64	0.61	0.64
Paper [118] Figure 5	0.45	0.60	0.42	0.50	0.43	0.55
Paper [118] Figure 6	0.58	0.67	0.73	0.80	0.65	0.73
Paper [83] figure 5	0.82	1.00	0.90	1.00	0.86	1.00
Paper [79] figure 1	0.22	0.64	0.29	0.67	0.25	0.65
Paper [79] figure 3	0.44	0.86	0.57	0.86	0.50	0.86
Paper [117] figure 1B	0.40	0.86	0.57	0.86	0.47	0.86
Paper [117] figure 2	0.61	0.55	0.33	0.17	0.43	0.26
Paper [117] figure 3	0.30	0.78	0.47	0.88	0.37	0.82
Paper [117] figure 11	0.83	1.00	0.77	0.85	0.80	0.92
Paper [117] figure 19	0.83	0.90	0.91	0.82	0.87	0.86
Paper [117] figure 20	0.67	0.88	0.67	0.78	0.67	0.82

Table 6. Precision, recall, andf-measure obtained by our tool with and without shape detection (SD). This table shows the results for those models that presented different results for the used approached.

19 models (seven Matlab Simulink models, and 12 models from the scientific papers), and the recall in 18 models (seven Matlab Simulink models, and 11 models from the scientific papers).

We hypothesize that XAMÃ with shape detection feature outperforms the version without the shape detection in terms of both precision and recall. Formally, we state the following hypotheses:

- H_0^p: The median difference between the *precision* for XAMÃ with and without shape detection is zero.
- H_a^p: The median difference between the *precision* for XAMÃ with and without shape detection is greater than zero.
- H_0^r: The median difference between the *recall* for XAMÃ with and without shape detection is zero.
- H_a^r: The median difference between the *recall* for XAMÃ with and without shape detection is greater than zero.

To test these hypotheses we perform two paired Wilcoxon signed-rank tests, one for precision and another one for recall. Since we perform repeated tests we need to adjust the p-values to control for the false discovery rate. We use the method proposed by Benjamini and Yekutieli [78]. The adjusted p-values obtained for precision and recall are 3.2×10^{-5} and 7.3×10^{-3}, respectively. Hence, we can reject H_0^p and H_0^r and state that XAMÃ with shape detection outperforms XAMÃ without shape detection.

Models	Precision	Recall
UML	-	-
Matlab Simulink	SD	SD
Scientific papers	SD	SD

Table 7. Improvement approach that presents statistically better results organized by the domain. The "-" means inconclusive result, "SD" means with shape detection

We also investigated the corresponding hypotheses separately for UML diagrams, Matlab Simulink models, and models from scientific papers, using the same Wilcoxon signed-rank tests. The adjusted p-values are below the commonly used threshold of 0.05 for four of the six comparisons (three types of models $\times$ precision or recall). As conclusion the use of shape detection improved the results for Matlab Simulink models and models from the scientific papers in terms of precision and recall. For UML diagrams the results were inconclusive as presented in Table 7.

As expected, adding the shape detection on top of the heuristics represents additional costs. Therefore, we also calculated the time spent to execute each approach. For every model, we run our approaches 40 times, we exclude the first 10 results and we calculate the median. The results are presented in appendix Tables 12, 13, and 14. Finally, we use the medians to calculate the ratio that is presented in Figure 10. Our experiments were executed on a MacBook Pro 16” (2019) with an Intel 2.4-GHz 8-core i9 processor turbo boost 5.0 GHz, 32 GB 2666-MHz DDR4, AMD Radeon Pro 5300M 4 GB, Intel UHD Graphics 630 1536 MB, MacOS Catalina.

Even though the shape detection makes the tool 111.28 (median) times slower, they still present reasonable execution times. For example, Figure 3 from the paper by Zhou et al. [117] presents the slowest execution time (with shape detection) around 0.02 seconds. Therefore, using shape detection not only produce better results but also performs reasonably well.

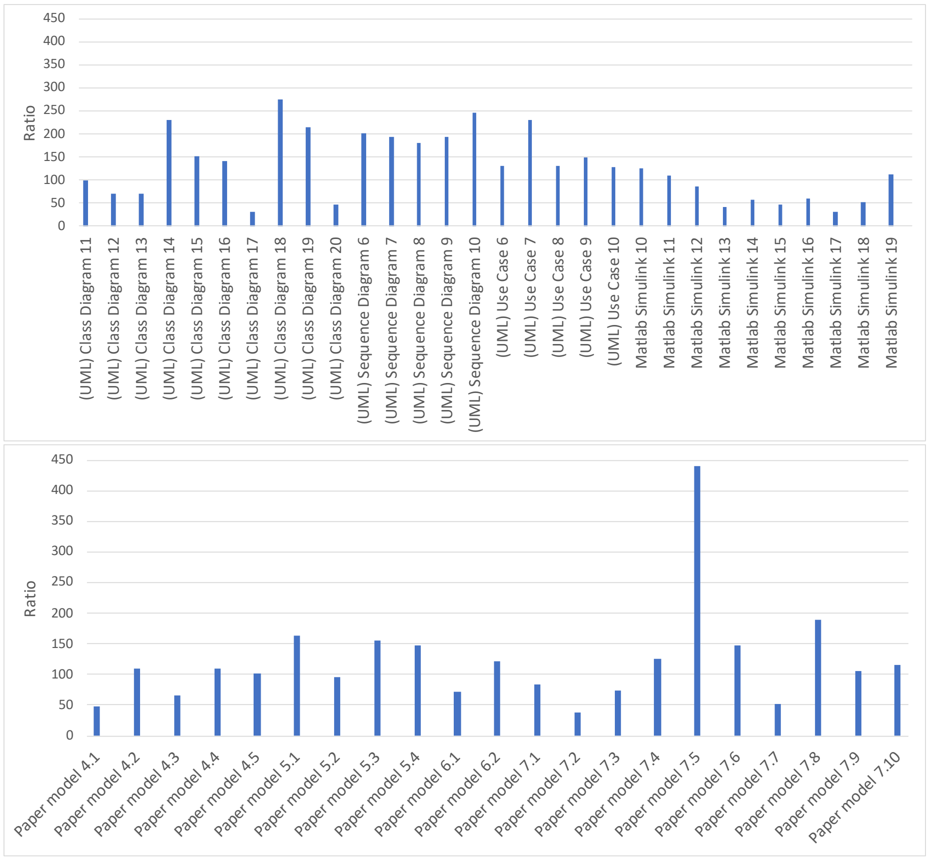

Fig. 10. Presents the ratio between the two approaches (with and without shape detection). The ratio is calculated using the median execution time using shape detection, divided by the median execution time without the shape detection. The mapping of paper model ID and the models from the selected scientific papers is presented in the Appendix Table 11

Take away message

XAMÃ correctly (without/with shape detection feature) identified 905/956 out of 1171 elements, presenting precision of 75/84%, recall of 77/82%, and f-measure of 76/83%. With shape detection XAMÃ produced statistically better results for Matlab Simulink models and model from scientific papers.

RQ4 - What is the overall impact, in terms of precision and recall, of using our approaches? To answer this research question, we compared the results produced by Google Cloud Vision with the results produced by XAMÃ with and without shape detection. For the following analyzes, even if the identification and merging of the text fragment positioned on multiple lines is made correctly, the "final" version of the text might be wrong due to another kind of error such as *missing char* or *extra char*. For example, assuming that the OCR service, wrongly, recognizes the element "Application Controller" from Figure 6 as two elements "Applicati0n" and "Contr0ller". By applying XAMÃ, these elements become "Applicati0n Contr0ller". Even though the merging is correct, the text fragment is wrong because of other kind of error (*character confusion*). Therefore, a priori the correction of the *multi-line text* error might not lead to higher precision and recall in general.

Without Shape Detection We observe the same results in 27 out of 51 models. XAMÃ presents better results in 22 models in terms of precision and f-measure, and in 16 models in terms of recall. Google Cloud Vision presents better results in one model in terms of precision, and in two models in terms of recall and f-measure. The overall precision and recall are presented in Table 8.

As expected, the overall improvements are better noticed on those models that present more text fragments positioned on multiple lines. The best improvements on precision, recall, and f-measure are on the following models:

– Figure 2 from the paper by Dounis et al. [83] with values improving from 13% to 50% (precision), 9% to 27% (recall) and 11% to 35% (f-measure);
– Figures 1b and 3 from the paper by Zhou et al. [117] with values improving from 17% to 40% (precision), 29% to 57% (recall) and 21% to 47% (f-measure) for Figure 1b, and from 2% to 30% (precision), 3% to 47% (recall) and 2% to 37% (f-measure) for figure 3.

We hypothesize that XAMÃ will outperform Google Cloud Vision in terms of both precision and recall. Formally, we state the following hypotheses:

– H_0^p: The median difference between the *precision* for XAMÃ and Google Cloud Vision is zero.
– H_a^p: The median difference between the *precision* for XAMÃ and Google Cloud Vision is greater than zero.

Model	Precision		Recall		F-Measure	
	GCV	Our Tool (H/SD)	GCV	Our Tool (H/SD)	GCV	Our Tool (H/SD)
Class Diagram (UML) 13	0.82	0.77	0.85	0.79	0.84	0.78
Class Diagram (UML) 14	0.75	0.75/0.73	0.75	0.75/0.69	0.75	0.75/0.71
Class Diagram (UML) 15	0.85	0.85/0.80	0.85	0.85/0.77	0.85	0.85/0.78
Class Diagram (UML) 16	0.68	0.72	0.71	0.74	0.69	0.73
Class Diagram (UML) 17	0.59	0.74	0.66	0.70	0.62	0.72
Class Diagram (UML) 20	0.49	0.49	0.43	0.40	0.46	0.44
Sequence Diagram (UML) 9	0.74	0.83	0.82	0.88	0.78	0.86
Sequence Diagram (UML) 10	0.65	0.68	0.81	0.81	0.72	0.74
Use Case (UML) 8	0.67	0.69	0.80	0.80	0.73	0.74
Matlab Simulink 10	0.78	0.78/0.94	0.88	0.88/0.94	0.82	0.82/0.94
Matlab Simulink 11	0.25	0.25/0.43	0.29	0.29/0.43	0.27	0.27/0.43
Matlab Simulink 13	0.33	0.43	0.37	0.43	0.35	0.43
Matlab Simulink 14	0.43	0.72/0.91	0.59	0.59/0.91	0.50	0.65/0.91
Matlab Simulink 15	0.14	0.18/0.24	0.21	0.26/0.32	0.17	0.21/0.27
Matlab Simulink 16	0.60	0.70	0.58	0.58	0.59	0.63
Matlab Simulink 17	0.68	0.73/0.79	0.81	0.85/0.88	0.74	0.79/0.84
Matlab Simulink 18	0.67	0.74/0.82	0.80	0.85/0.90	0.73	0.79/0.86
Matlab Simulink 19	0.89	0.89/1	0.94	0.94/1	0.92	0.92/1
Paper [118] Figure 1	0.23	0.36/0.68	0.43	0.57/0.81	0.30	0.44/0.74
Paper [118] Figure 2	0.33	0.33/0.36	0.36	0.36/0.36	0.35	0.35/0.36
Paper [118] Figure 5	0.18	0.18/0.30	0.17	0.17/0.25	0.17	0.17/0.27
Paper [118] Figure 6	0.16	0.22/0.28	0.20	0.27/0.33	0.18	0.24/0.30
Paper [83] Figure 2	0.13	0.50	0.09	0.27	0.11	0.35
Paper [83] Figure 5	0.73	0.73/0.90	0.80	0.80/90	0.76	0.76/90
Paper [79] Figure 1	0.11	0.21/0.60	0.17	0.29/0.63	0.13	0.25/0.61
Paper [79] Figure 3	0.42	0.44/0.86	0.57	0.57/0.86	0.48	0.50/0.86
Paper [117] Figure 1b	0.17	0.40/0.75	0.29	0.57/0.86	0.21	0.47/0.80
Paper [117] Figure 2	0.45	0.54/0.48	0.29	0.30/0.23	0.35	0.39/0.31
Paper [117] Figure 3	0.02	0.30/0.78	0.03	0.47/0.88	0.02	0.37/0.82
Paper [117] Figure 4	0.27	0.31	0.50	0.50	0.35	0.38
Paper [117] Figure 11	0	0.08/0.18	0	0.08/0.15	0	0.08/0.17
Paper [117] Figure 19	0.69	0.83/0.90	0.82	0.91/0.82	0.75	0.87/0.86
Paper [117] Figure 20	0.67	0.67/0.88	0.67	0.67/0.78	0.67	0.67/0.82

Table 8. Precision, recall, and f-measure obtained by Google Cloud Vision (GCV) and our tool (S/SD) where "H" means without shape detection, and "SD" means with shape detection.

- H_0^r: The median difference between the *recall* for XAMÃ and Google Cloud Vision is zero.
- H_a^r: The median difference between the *recall* for XAMÃ and Google Cloud Vision is greater than zero.

To test these hypotheses we perform two paired Wilcoxon signed-rank tests, one for precision and another one for recall, as described previously in RQ3.

The adjusted p-values obtained for precision and recall are 5.7×10^{-6} and 2.3×10^{-4}, respectively. Hence, we can reject H_0^p and H_0^r and state that XAMÃ without shape detection outperforms Google Cloud Vision.

Models	Precision (H/SD)	Recall (H/SD)
UML	-/-	-/-
Matlab Simulink	H/SD	-/SD
Scientific papers	H/SD	-/SD

Table 9. OCR service that presents statistically better results organized by the domain. The "-" means inconclusive result, "H" means without shape detection, and "SD" means with shape detection

This discussion indicates that while overall XAMÃ outperforms Google Cloud Vision, this does not imply that this should also be the case for models of different domains. This is why we formulate the corresponding hypotheses separately for UML diagrams, Matlab Simulink models, and models from scientific papers. They were tested using the same paired Wilcoxon signed-rank tests as before.

The adjusted p-values are below the commonly used threshold of 0.05 for two of the six comparisons (three types of models $\times$ precision or recall). As conclusion XAMÃ improved the results for Matlab Simulink models and models from the scientific papers in terms of precision. For UML diagrams the results were inconclusive, as presented in Table 9.

With Shape Detection We observe that XAMÃ presents the same results of Google Cloud Vision in terms of both precision and recall in 19 models. XAMÃ presents better result in terms of precision in 29 models, and recall in 22 models. Google Cloud Vision presents better results in three models in terms of precision, and in five models in terms of recall and f-measure. The best improvements on precision, recall, and f-measure are on the following models:

- Figure 1 from the paper by Blasera et al. [79] with values improving from 11% to 60% (precision), 17% to 63% (recall) and 13% to 61% (f-measure);
- Figures 1b and 3 from the paper by Zhou et al. [117] are also presented as the best improvements when using XAMÃ without the shape detection feature. However, using shape detection XAMÃ could improve the results

even further. For Figure 1b the precision is 75%, recall is 86% , and f-measure is 80%. For Figure 3 the precision is 78%, recall is 88% and f-measure is 82% .

Since XAMÃ without the shape detection feature outperformed Google Cloud Vision, we expected that XAMÃ with the shape detection feature would also outperform Google Cloud Vision. Therefore, we tested these hypotheses using the same two paired Wilcoxon signed-rank tests. As result, the tests confirms our hypothesis presenting adjusted p-values of 1.3×10^{-7} and 3.1×10^{-5} for precision and recall respectively.

We also tested these hypotheses separately for UML diagrams, Matlab Simulink models, and models from scientific papers. Using the shape detection feature, XAMÃ improved the results for Matlab Simulink models and models from scientific papers, but for UML models the tests presents inconclusive results. It is worth mentioning that the amount of text fragments positioned on multiple lines, causing the *multi-line text* error, is not high on UML diagrams (17 out of 604 elements). Therefore, there is not enough *multi-line text* error in the UML diagrams to be correct by XAMÃ that could lead to statistically better results. The distribution of these elements that could cause the *multi-line text* error is the following: nine elements in three class diagrams, three elements in two sequence diagrams, and five in two use cases.

Take away message

With shape detection feature XAMÃ outperforms Google Cloud Vision in terms of precision and recall for Matlab Simulink models and models from scientific papers. Without the shape detection feature, XAMÃ outperforms Google Cloud Vision only in terms of precision. The results are inconclusive for UML diagrams due to the low amount of *Multi-line text* error that could be corrected by XAMÃ.

RQ5 - How accurate is XAMÃ compared to a domain specific tool? We investigate how XAMÃ which is a combination of a general-purpose OCR technique and our generic approach (with and without shape detection features) would perform compared to a domain specific tool. To answer this question, we select img2UML as a domain specific tool because of its accuracy in extracting UML models from images [96].

Img2UML [95, 96] is a system written in VB.NET and using MODI library to extract the text from images. The system takes an image of a class diagram as input, and as output it generates a XMI file that represents the class diagram to be visualized by StarUML CASE tool.

Since Img2UML has been designed specifically for UML class diagrams, we compare the precision and recall only for this type of models. To answer this research question, we analyzed 20 UML class diagrams. The UML class diagrams used to answer RQ1 and RQ2, and additional 10 UML class diagrams [34–42, 44].

It is worth mentioning that we only compare the accuracy related to the text recognition. The classes and relationships identification are out of the scope of this study.

In overall, XAMÃ correctly recognized 433/431 (without/with shape detection) out of 614 elements, while Img2UML correctly recognized 171 elements. Table 10 presents the precision, recall, and f-measure obtained for these class diagrams. We observe that XAMÃ presented better results than Img2UML in all cases. Therefore, we hypothesize that XAMÃ outperforms Img2UML in terms of both precision and recall. Formally, we state the following hypotheses:

– H_0^p: The median difference between the *precision* for XAMÃ and img2UML is zero.
– H_a^p: The median difference between the *precision* for XAMÃ and img2UML is greater than zero.
– H_0^r: The median difference between the *recall* for XAMÃ and img2UML is zero.
– H_a^r: The median difference between the *recall* for XAMÃ and img2UML is greater than zero.

To evaluate these hypotheses, we used the same statistic tests as before. XAMÃ with or without shape detection feature does not present significantly different values for precision and recall on UML class diagrams. Therefore, we expect that the p-values obtained from the tests between img2UML and XAMÃ with shape detection, and between img2UML and XAMÃ without shape detection would be the same. The adjusted p-values obtained for both precision and recall is 1.4×10^{-6}. Hence, we can reject H_0^p and H_0^r and state that XAMÃ outperforms img2UML.

We also calculate the Cliff's delta [81]. A non-parametric effect size measure for ordinal data. We consider the effect size values: $|d| < 0.147$ "negligible", $|d| < 0.33$ "small", $|d| < 0.474$ "medium", otherwise "large" as defined by Romano et al. [108]. We obtained a large Cliff's delta value -0.8625, and a 95% confidence interval being between -0.98 and -0.25.

Take away message

XAMÃ outperforms Img2UML in terms of precision and recall being capable of correctly recognize 433/431 (without/with shape detection) out of 614 elements, as oppose to 171 by Img2UML.

4 Threats to Validity

As any empirical study our work is subject to threats to validity. Wohlin et al. [116] provide a list of possible threats that researchers can face during a scientific research. In this section, we describe the actions we took in order to increase the validity and decrease the threats.

	Precision		Recall		F-Measure	
Model	Img2UML	XAMÃ (H/SD)	Img2UML	XAMÃ (H/SD)	Img2UML	XAMÃ (H/SD)
CD 11	0.55	0.85	0.35	0.87	0.43	0.86
CD 12	0.63	0.80	0.50	0.80	0.56	0.80
CD 13	0.50	0.77	0.47	0.79	0.49	0.78
CD 14	0.31	0.75/0.73	0.31	0.75/0.69	0.31	0.75/0.71
CD 15	0.38	0.85/0.80	0.38	0.85/0.77	0.38	0.85/0.78
CD 16	0.08	0.72	0.08	0.74	0.08	0.73
CD 17	0.00	0.74	0.00	0.70	0.00	0.72
CD 18	0.13	0.96	0.13	0.96	0.13	0.96
CD 19	0.65	0.82	0.65	0.82	0.65	0.82
CD 20	0.08	0.49	0.05	0.40	0.06	0.44
CD 21	0.50	0.56	0.42	0.53	0.46	0.54
CD 22	0	0.67	0	0.80	0	0.73
CD 23	0.70	0.96	0.58	0.96	0.64	0.96
CD 24	0.22	0.61/0.63	0.19	0.65	0.20	0.63/0.64
CD 25	0.29	0.94	0.29	0.94	0.29	0.94
CD 26	0.13	0.60/0.71	0.14	0.64/0.71	0.14	0.62/0.71
CD 27	0.44	0.61	0.42	0.60	0.43	0.61
CD 28	0.06	0.53	0.05	0.53	0.05	0.53
CD 29	0.38	0.50	0.38	0.50	0.38	0.50
CD 30	0.42	0.67	0.42	0.67	0.42	0.67

Table 10. Precision, recall, and f-measure obtained by Img2UML and XAMÃ. The "H" means without shape detection, and "SD" means with shape detection.

Internal validity, concerns the unknown influences of independent variables can have on studies. In order to mitigate this concern, we have selected OCR services that have been evaluated by previous studies on different text recognition tasks. While the manual extraction of textual elements has been performed by one author only, the task is simple for an engineer and is unlikely to be affected by the subjectivity of their judgment. Another threat is that we do not know which version of Google Cloud Vision we used, nor we can chose the version. As consequence it is possible that future Google Cloud Vision update might break our heuristics. In order to mitigate this problem, we opted to be as conservative as possible regarding the size of the tolerance range used to identify whether the text is positioned on multiples lines or not.

External validity concerns the generalizability of the results and findings of the study. In order to mitigate this concern, we have diversified the collection of models analyzed to include models from different domains and different sources.

Construct validity concerns the issues related to the design of the experiment. In order to address this issue, we used metrics that were sufficiently defined in previous studies. Example of such metrics are precision, recall, and F-measure. We used these metrics to indicate which OCR service presents better performance.

Conclusion validity concerns about the relations between the conclusions that we draw and the analyzed data. In order to mitigate this concern, we paid special attention to use appropriate statistical techniques, and we described all decisions we made. Thus, this study can be replicated by other researchers, and we expect our results to be quite robust.

5 Discussion and Future Work

The results described in this paper can serve as a starting point for future research on the use of OCR for multi-domain model management, as well as for design of tools supporting multi-domain model management. Our expectation is that OCR can be used to support the multi-domain model management by identifying the relationships between models from different domains. In particular in those scenarios where only the images of the models are available, or in those scenarios where the source code of the models are available but there is no communication between the modeling tools.

We started by investigating accuracy of the off-the-shelf OCR services for extracting text from graphical models. Concurrent with the previous studies [2,107] Google Cloud Vision outperformed Microsoft Cognitive Services on both precision and recall. However, the precision and recall values of Google Cloud Vision were not as high as the ones presented in the previous studies [2]. We believe this is due to the difference between the analyzed items: graphical models vs. business names. As opposed to business names, graphical models often include mathematical elements such as Greek letters and subscripts, and non-alphanumeric characters. Moreover, extracting text from models that do not follow the same design rules, incurs additional challenges. Indeed, the precision and recall scores for models from scientific papers are much more spread out in Figure 2, than for models from other data sources.

Next, we investigated the common errors produced by Google Cloud Vision. We identified 17 different types of errors organized by four categories: *non-alphanumeric characters*, *mathematical formulas*, *spacing*, and *character confusion*. Most common errors are related to Spacing and Character confusion; however, the main challenges seem to be related to the *multi-line text* error and mathematical formulas—not a single Greek letter, subscript or equation appearing in the models could be correctly identified. Therefore, we proposed two approaches to correct the *multi-line text* error. To achieve this goal, we investigated a set of models that have this problem and we defined a collection of heuristics to be used to correct them. The main challenge in our approach was to identify right heuristics that would present a good accuracy independently of the domains of the model in study. Hence, we opted to be as conservative as possible. One approach consists using the position (coordinates) of the text fragments, and the other approach adds shape detection feature to group the text fragments improving the results.

We evaluated the accuracy of XAMÃ in identifying whether the text fragment is positioned on multiple lines and if the merging of the text fragments is performed correctly. In this evaluation we ignored other error types that XAMÃ could not fix, such as *character confusion*. We observed that XAMÃ, without the shape detection feature, correctly identified 905 out of 1171 elements positioned on single or multiple lines. We consider these numbers as good results, specially because of the fact it is a lightweight approach that uses the coordinates to identify whether the text fragments should be merged or not. We improved these numbers when we combined a slightly modified version of these heuristics with the identification of shapes such as boxes. Using the shape detection feature the tool correctly identified 956 elements and improved the precision from 75% to 84%, the recall from 77% to 82% and f-measure from 76% to 83%. We evaluate this as good improvements and they were statistically better on Matlab Simulink and models retrieved from scientific papers.

As expected, the combination of these techniques demands more computational processing making the tool slower. Thus, we calculated the costs (time) of using the shape detection, and we observed that this feature makes the tool 111.28 (median) times slower. Although this number might represent a high ratio, in practice it is not noticeable for human beings because the measurement unit of time is nanosecond. Having the slowest execution time as example. It took 0.02 seconds to be executed representing negligible costs. Finally, we conclude that using the shape detection feature improve the text recognition and presents a reasonable execution time. It is worth mentioning that this feature was used only to identify whether the text fragments are positioned on multiple lines or not. The reason is because this paper is mainly about the text extraction and how to improve the OCR results. For future work, we intend to use shape detection features to extract semantics from the models.

Next, we compared the overall accuracy between XAMÃ and Google Cloud Vision. We performed two paired Wilcoxon signed-rank tests to conclude that our approaches outperformed Google Cloud Vision. As expected the accuracy presented by XAMÃ was higher on those models that have more text fragments positioned on multiple lines. This analysis shows the importance of correcting this error. By having the models from scientific papers as example: we observed that in Figure 2 from the paper by Dounis et al. [83] the precision increased almost 40%. We obtained even better improvements when using the shape detection, for instance in Figure 3 from the paper by Zhou et al. [117] the improvement on precision was 76%, and 84% on recall.

The imprecision of the coordinates provided by Google Cloud Vision partially explains the reason why some models present low precision and recall. Due to this imprecision, we had to add a tolerance range when evaluating whether the text fragment is positioned on multiple lines or not. When increasing the tolerance range, we also increase the number of identified text positioned on multiple lines. However, it causes a raise of false positives due to the misinterpretation of text that are not positioned on multiple lines. We decided not to increase the tolerance range because we opted for high precision to the detriment of recall.

Finally, we compared the results produced by XAMÃ with the ones presented by img2UML, a domain specific tool. We also performed two paired Wilcoxon signed-rank tests and we concluded that XAMÃ outperformed img2UML both in precision and recall. Text recognition is only one part of img2UML, this tool is also capable of detecting shapes in order to identify the classes and the relationships between the classes. In this evaluation we only compared the text recognition. We conjecture two possible reasons for the weak accuracy by img2UML. The first reason might be the fact img2UML uses an outdated OCR engine for text recognition. Hence, we believe that img2UML can improve significantly once it updates the OCR engine to a newer one such as Google Cloud Vision. The second reason might be that img2UML is optimized to class diagrams exported by one specific CASE tool.

As future work, we intend to focus on the main challenges we identified in Section 2.2. We investigated the possibility of using OCR to extract text from models from different domains, and we proposed two approaches to correct the *multi-line text* error produced during the text extraction. However, all analyzed graphical models were designed using computer tools, and we believe that extracting information from hand-written model also plays an important role in the model management. Furthermore, we want to evaluate different OCR techniques and our approaches on additional kinds of graphical models, including, for instance, SysML models, models drawn on whiteboards and hand-written models. We also want to evaluate if combining XAMÃ with Bayesian algorithm [98] can improve the text recognition, in particular fixing the multi-line text error. Simultaneously, we intend to combine OCR with image processing to analyze additional graphical elements such as lines, and arrows presented in the models. There are some additional research questions that need to be answered for a better evaluation of OCR as technology to support the automatic identification of relationships. These questions include: what are the consequences of a missed recognition by the OCR tool? How much effort is needed to identify errors by the OCR tool? We also want to evaluate the accuracy of using only the text extracted from the models to identify possible relationships. In parallel, we can compare the results of detecting possible relationships with/without semantics detection.

6 Related Work

To the best of our knowledge, there are no studies on the use of off-the-shelf OCR services on models from different domains, as well as no studies proposing techniques to correct *multi-line text* errors. There are, however, a number of studies in which OCR has been applied to domain-specific models, and studies proposing post possessing techniques to correct misspellings produced by OCR. These studies are described below:

Img2UML [95, 96] extracts UML Class Diagrams from images, identifying, e.g., class names, fields and methods. Img2UML uses Microsoft Office Document Imaging as the OCR technique for text recognition. While Img2UML is geared towards and evaluated on a specific domain, the techniques we have analyzed have been applied to models of multiple domains. Several studies have used OCR as part of a tool classifying images as UML diagrams: targeting class diagrams [92, 102], sequence diagrams [106] and component diagrams [102].

Going beyond engineering models, Reis [107] compare Google Cloud Vision and Microsoft Cognitive Services in recognizing text from the photos of the pages of the Bible. Additional comparison studies have been published by Mello and Dueire Lins [99] and Vijayarani and Sakila [115].

Regarding post possessing techniques, Cacho et. al. [80] propose combining edit Levenshtein distance with the concept of context given by trigrams i.e. a contiguous sequence of three words. The authors use Google Web 1T Database as context to support the acceptance of words with bigger edit Levenshtein distance in case where the candidate word makes more sense in the context of the sentence.

Bassil and Alwani [76] propose using the Google online spelling suggestion to identify and correct words that have been misspelled during the OCR process. When applied this approach, the authors observed a significant improvement in OCR error correction rate.

Kanjanawattana and Kimura [94] propose an approach that uses ontologies, natural language processing (NPL), and edit distance to improve the OCR result correction. Differently from the other approaches, in this study the authors use a collection of two-dimensional bar graphs from journal articles as dataset. The authors concluded that their solution is effective because of the high accuracy compared to other methods.

Lasko and Hauserb [98] evaluated five methods to correct misspelled words during the OCR process. The methods evaluated in this study were: the edit distance algorithm with a probabilistic substitution matrix, an adaptation of the cross correlation algorithm, Bayesian analysis, Bayesian analysis on an actively thinned reference dictionary, and the generic edit distance algorithm. The authors concluded that Bayesian algorithm is the most accurate one.

Khirbat [97] proposes the use of support vector machine (SVM) to identify misspellings, and the use of Levenshtein's edit distance and simulated annealing (SA) algorithm to correct the words. In the end they validate the words checking whether they are present in Google Web n-gram corpus.

Perianez-Pascual et. al. [104] evaluate the use of OCR in recognizing OCL expressions and propose strategies to improve the quality of this recognition. They evaluate these strategies in a collection of 3250 images of OCL expressions. As conclusion, they improve the OCR recognition in 15.63%.

7 Conclusion

We presented a study of suitability of the off-the-shelf OCR services in the context of multi-domain model management. We evaluated performance of two well-known services, Google Cloud Vision and Microsoft Cognitive Services, on a collection of 43 models from different domains: 17 UML diagrams, 9 MatLab Simulink models and 17 models from scientific papers from the control system engineering domain.

We observed that Google Cloud Vision overall outperforms Microsoft Cognitive Services both in terms of precision and in terms of recall. This observation is consistent both with the previous work [2,107] and with a follow-up study investigating performance of the two OCR-services on models of different domains.

Focusing on Google Cloud Vision, we identified a list of 17 kinds of errors distributed over four categories: non-alphanumeric characters, mathematical formulas, spacing and character confusion. Among these errors, the most common are related to text written on multiple lines, wrong/missing characters, and an empty space between letters. It is also important that in presence of multi-line texts, Greek letters, subscripts, and equations because Google Cloud Vision failed every single time.

We presented XAMÃ, a tool that includes two approaches to correct the *multi-line text* error by identifying and merging textual elements that are positioned on multiple lines. We evaluated these approaches in a collection of 51 models from different domains: 20 UML diagrams, 10 MatLab Simulink models, and 21 models from scientific papers from the system engineering domain. For this goal, we defined a set of heuristics that take into consideration the location, alignment, and the distance of the text fragments. For the second approach we combined a slightly modified version of these heuristics with a shape detection feature using well-known image process techniques.

We compared these two approaches in terms of precision, recall, and f-measure, and we concluded that XAMÃ, with shape detection feature, presents statistically better results on Matlab Simulink and models from scientific papers. We also compared out tool with Google Cloud Vision, and XAMÃ outperformed Google Cloud Vision.

Additionally, we compared the results produced by XAMÃ with the ones presented by img2UML. We observed that XAMÃ outperformed img2UML both in precision and recall. We conjecture that the reason might be due the use of an outdated OCR engine.

To conclude, we observed that correcting *multi-line text* error can increase significantly the overall accuracy, specially when the model presents a high number of text positioned on multiple lines. XAMÃ produces produces goods results on both approaches (with and without shape detection).

Acknowledgements

Research leading to these results has received funding from the EU ECSEL Joint Undertaking under grant agreement n° 826452 (project Arrowhead Tools) and from the partners national programs/funding authorities.

References

1. Git uml repository. https://www.gituml.com. Accessed: 2020-01-23
2. Image text recognition apis showdown. https://dataturks.com/blog/compare-image-text-recognition-apis.php. Accessed: 2020-01-08
3. Matlab simulink model 1. https://nl.mathworks.com/help/simulink/slref/anti-windup-control-using-a-pid-controller.html. Accessed: 2020-01-24
4. Matlab simulink model 10. https://nl.mathworks.com/help/simulink/slref/bumpless-control-transfer-between-manual-and-pid-control.html. Accessed: 2020-08-06
5. Matlab simulink model 11. https://nl.mathworks.com/help/simulink/slref/vehicle-electrical-system.html. Accessed: 2020-08-06
6. Matlab simulink model 12. https://nl.mathworks.com/help/simulink/slref/vehicle-electrical-and-climate-control-systems.html. Accessed: 2020-08-06
7. Matlab simulink model 13. https://nl.mathworks.com/help/simulink/slref/aircraft-longitudinal-flight-control.html. Accessed: 2020-08-06
8. Matlab simulink model 14 and 15. https://nl.mathworks.com/help/simulink/slref/designing-a-high-angle-of-attack-pitch-mode-control.html. Accessed: 2020-08-06
9. Matlab simulink model 16. https://nl.mathworks.com/help/simulink/slref/friction-model-with-hard-stops.html. Accessed: 2020-08-06
10. Matlab simulink model 17. https://nl.mathworks.com/help/simulink/slref/thermal-model-of-a-house.html. Accessed: 2020-08-06
11. Matlab simulink model 18. https://nl.mathworks.com/help/simulink/slref/inverted-pendulum-with-animation.html. Accessed: 2020-08-06
12. Matlab simulink model 19. https://nl.mathworks.com/help/simulink/slref/tank-fill-and-empty-with-animation.html. Accessed: 2020-08-06
13. Matlab simulink model 2. https://nl.mathworks.com/help/simulink/slref/simulating-automatic-climate-control-systems.html. Accessed: 2020-01-24
14. Matlab simulink model 3. https://nl.mathworks.com/help/simulink/slref/simulation-of-a-bouncing-ball.html. Accessed: 2020-01-24
15. Matlab simulink model 4. https://bit.ly/simulinkModel4. Accessed: 2020-01-24
16. Matlab simulink model 5. https://bit.ly/simulinkModel5. Accessed: 2020-01-24
17. Matlab simulink model 6. https://bit.ly/simulinkModel6. Accessed: 2020-01-24

18. Matlab simulink model 7. https://bit.ly/simulinkModel7. Accessed: 2020-01-24
19. Matlab simulink model 8. https://nl.mathworks.com/help/simulink/slref/designing-a-guidance-system-in-matlab-and-simulink.html. Accessed: 2020-01-24
20. Matlab simulink model 9. https://bit.ly/simulinkModel9. Accessed: 2020-01-24
21. Opencv (open source computer vision library). https://opencv.org. Accessed: 2021-1-19
22. Uml - class diagram 1. http://models-db.com/repository/70/classdiagram/238. Accessed: 2020-01-24
23. Uml - class diagram 11. https://www.gituml.com/viewz/30. Accessed: 2020-08-06
24. Uml - class diagram 12. https://www.gituml.com/viewz/1. Accessed: 2020-08-06
25. Uml - class diagram 13. https://www.gituml.com/viewz/100. Accessed: 2020-08-06
26. Uml - class diagram 14. http://models-db.com/repository/76/classdiagram/367. Accessed: 2020-08-06
27. Uml - class diagram 15. http://models-db.com/repository/76/classdiagram/370. Accessed: 2020-08-06
28. Uml - class diagram 16. http://models-db.com/repository/104/classdiagram/681. Accessed: 2020-08-06
29. Uml - class diagram 17. http://models-db.com/repository/104/classdiagram/685. Accessed: 2020-08-06
30. Uml - class diagram 18. http://models-db.com/repository/100/classdiagram/619. Accessed: 2020-08-06
31. Uml - class diagram 19. http://models-db.com/repository/70/classdiagram/233. Accessed: 2020-08-06
32. Uml - class diagram 2. https://www.gituml.com/viewz/5. Accessed: 2020-01-24
33. Uml - class diagram 20. http://models-db.com/repository/142/classdiagram/1391. Accessed: 2020-08-06
34. Uml - class diagram 21. https://www.gituml.com/viewz/29. Accessed: 2021-07-15
35. Uml - class diagram 22. https://www.gituml.com/viewz/194. Accessed: 2021-07-15
36. Uml - class diagram 23. https://www.gituml.com/viewz/313. Accessed: 2021-07-15
37. Uml - class diagram 24. https://www.gituml.com/viewz/25. Accessed: 2021-07-15
38. Uml - class diagram 25. http://models-db.com/repository/82/classdiagram/376. Accessed: 2021-07-15
39. Uml - class diagram 26. http://models-db.com/repository/104/classdiagram/679. Accessed: 2021-07-15
40. Uml - class diagram 27. http://models-db.com/repository/84/classdiagram/440. Accessed: 2021-07-15
41. Uml - class diagram 28. http://models-db.com/repository/84/classdiagram/453. Accessed: 2021-07-15
42. Uml - class diagram 29. http://models-db.com/repository/84/classdiagram/471. Accessed: 2021-07-15
43. Uml - class diagram 3. https://www.gituml.com/viewz/87. Accessed: 2020-01-24
44. Uml - class diagram 30. http://models-db.com/repository/84/classdiagram/472. Accessed: 2021-07-15
45. Uml - class diagram 4. https://www.gituml.com/viewz/26. Accessed: 2020-01-24
46. Uml - class diagram 5. https://www.gituml.com/viewz/27. Accessed: 2020-01-24
47. Uml - class diagram 6. https://www.gituml.com/viewz/20. Accessed: 2020-01-24
48. Uml - class diagram 7. http://models-db.com/repository/84/classdiagram/441. Accessed: 2020-01-24

49. Uml - class diagram 8. http://models-db.com/repository/84/classdiagram/449. Accessed: 2020-01-24
50. Uml - class diagram 9. http://models-db.com/repository/102/classdiagram/624. Accessed: 2020-01-24
51. Uml - sequence diagram 1. http://models-db.com/repository/108/classdiagram/781. Accessed: 2020-01-24
52. Uml - sequence diagram 10. https://raw.githubusercontent.com/paglian/QSimple Calc/ master/doc/SequenceDiagram.png. Accessed: 2020-09-22
53. Uml - sequence diagram 2. http://models-db.com/repository/108/classdiagram/783. Accessed: 2020-01-24
54. Uml - sequence diagram 3. http://models-db.com/repository/108/classdiagram/808. Accessed: 2020-01-24
55. Uml - sequence diagram 4. http://models-db.com/repository/108/classdiagram/809. Accessed: 2020-01-24
56. Uml - sequence diagram 6. http://models-db.com/repository/108/classdiagram/810. Accessed: 2020-08-06
57. Uml - sequence diagram 7. https://raw.githubusercontent.com/glindstrom/OhHa/ master/dokumentointi/SequenceDiagram1.png. Accessed: 2020-09-22
58. Uml - sequence diagram 8. https://raw.githubusercontent.com/glindstrom/OhHa/ master/dokumentointi/SequenceDiagram3.png. Accessed: 2020-09-22
59. Uml - sequence diagram 9. https://raw.githubusercontent.com/mcfa77y/python/ master/input/gliffy/create_new_customer_sequence_diagram.png. Accessed: 2020-09-22
60. Uml - use case diagram 1. http://models-db.com/repository/108/classdiagram/733. Accessed: 2020-01-24
61. Uml - use case diagram 10. http://models-db.com/repository/108/classdiagram/796. Accessed: 2020-08-06
62. Uml - use case diagram 2. http://models-db.com/repository/108/classdiagram/734. Accessed: 2020-01-24
63. Uml - use case diagram 3. http://models-db.com/repository/108/classdiagram/736. Accessed: 2020-01-24
64. Uml - use case diagram 4. http://models-db.com/repository/108/classdiagram/775. Accessed: 2020-01-24
65. Uml - use case diagram 6. http://models-db.com/repository/108/classdiagram/738. Accessed: 2020-08-06
66. Uml - use case diagram 7. http://models-db.com/repository/108/classdiagram/789. Accessed: 2020-08-06
67. Uml - use case diagram 8. http://models-db.com/repository/108/classdiagram/811. Accessed: 2020-08-06
68. Uml - use case diagram 9. http://models-db.com/repository/108/classdiagram/794. Accessed: 2020-08-06
69. The uml repository. http://models-db.com. Accessed: 2020-01-23
70. Omg unified modeling language (omg uml), v2.5.1. OMG Document Number formal/2017-12-05 (https://www.omg.org/spec/UML/About-UML/) (2007)
71. Ai, B., Sentis, L., Paine, N., Han, S., Mok, A., Fok, C.L.: Stability and performance analysis of time-delayed actuator control systems. Journal of Dynamic Systems, Measurement, and Control **138**(5), 1–20 (2016)
72. Akdur, D., Garousi, V., Demirörs, O.: A survey on modeling and model-driven engineering practices in the embedded software industry. Journal of Systems Architecture **91**, 62–82 (2018)

73. Akdur, D., Say, B., Demirörs, O.: Modeling cultures of the embedded software industry: Feedback from the field. Software and Systems Modeling **20**(2), 447–467 (2021)
74. Atkinson, C.: Orthographic software modelling: A novel approach to view-based software engineering. In: European Conference on Modelling Foundations and Applications, p. 1. Springer (2010)
75. Baltes, S., Diehl, S.: Sketches and diagrams in practice. FSE 2014, p. 530–541. Association for Computing Machinery, New York, NY, USA (2014)
76. Bassil, Y., Alwani, M.: Ocr post-processing error correction algorithm using google online spelling suggestion (2012)
77. Benjamini, Y., Hochberg, Y.: Controlling the False Discovery Rate: A Practical and Powerful Approach to Multiple Testing. Journal of the Royal Statistical Society. Series B (Methodological) **57**(1), 289–300 (1995)
78. Benjamini, Y., Yekutieli, D.: The control of the false discovery rate in multiple testing under dependency. Annals of statistics pp. 1165–1188 (2001)
79. Blaser, P., Pavliček, F., Mori, K., Mayr, J., Weikert, S., Wegener, K.: Adaptive learning control for thermal error compensation of 5-axis machine tools. Journal of Manufacturing Systems **44**, 302–309 (2017)
80. Cacho, J.R.F., Taghva, K., Alvarez, D.: Using the google web 1t 5-gram corpus for ocr error correction. In: International Conference on Information Technology-new Generations, pp. 505–511. Springer International Publishing, Cham (2019)
81. Cliff, N.: Dominance statistics: Ordinal analyses to answer ordinal questions. Psychological Bulletin **114**(3), 494–509 (1993)
82. Di Ruscio, D., Iovino, L., Pierantonio, A.: What is needed for managing co-evolution in mde? In: Proceedings of the 2nd International Workshop on Model Comparison in Practice, IWMCP '11, p. 30–38. Association for Computing Machinery (2011)
83. Dounis, A.I., Caraiscos, C.: Advanced control systems engineering for energy and comfort management in a building environment–a review. Renewable and Sustainable Energy Reviews **13**(6-7), 1246–1261 (2009)
84. Favre, J.M.: Languages evolve too! changing the software time scale. In: Eighth International Workshop on Principles of Software Evolution, pp. 33–42 (2005)
85. Feldmann, S., Kernschmidt, K., Wimmer, M., Vogel-Heuser, B.: Managing inter-model inconsistencies in model-based systems engineering: Application in automated production systems engineering. Journal of Systems and Software **153**, 105–134 (2019)
86. Feldmann, S., Wimmer, M., Kernschmidt, K., Vogel-Heuser, B.: A comprehensive approach for managing inter-model inconsistencies in automated production systems engineering. In: IEEE International Conference on Automation Science and Engineering, pp. 1120–1127. IEEE (2016)
87. Frechette, S.: Model based enterprise for manufacturing. In: Proceedings of the 44th CIRP international conference on manufacturing systems (2011)
88. Gonzalez, R.C., Woods, R.E.: Digital Image Processing. Prentice Hall (2002)
89. Hebig, R., Giese, H., Stallmann, F., Seibel, A.: On the complex nature of mde evolution. In: International Conference on Model Driven Engineering Languages and Systems, pp. 436–453. Springer (2013)
90. Hebig, R., Quang, T.H., Chaudron, M.R., Robles, G., Fernandez, M.A.: The quest for open source projects that use uml: Mining github. In: International Conference on Model Driven Engineering Languages and Systems, pp. 173–183 (2016)

91. Ho-Quang, T.: Empowering empirical research in software design: Construction and studies on a large-scale corpus of uml models. Ph.D. thesis, Department of Computer Science and Engineering (2019)
92. Ho-Quang, T., Chaudron, M.R., Samúelsson, I., Hjaltason, J., Karasneh, B., Osman, H.: Automatic classification of uml class diagrams from images. In: Asia-pacific Software Engineering Conference, vol. 1, pp. 399–406. IEEE (2014)
93. Kaliappan, V.K., Yong, H., Dugki, M., Choi, E., Budiyono, A.: Reconfigurable intelligent control architecture of a small-scale unmanned helicopter. Journal of Aerospace Engineering **27**(4), 1–13 (2014)
94. Kanjanawattana, S., Kimura, M.: Ontologies-based optical character recognition-error correction method for bar graphs. In: International Conference on Advances in Semantic Processing, pp. 1–8 (2016)
95. Karasneh, B., Chaudron, M.R.: Extracting uml models from images. In: International Conference on Computer Science and Information Technology, pp. 169–178. IEEE (2013)
96. Karasneh, B., Chaudron, M.R.: Img2uml: A system for extracting uml models from images. In: Euromicro Conference on Software Engineering and Advanced Applications, pp. 134–137. IEEE (2013)
97. Khirbat, G.: Ocr post-processing text correction using simulated annealing (opteca). In: Australasian Language Technology Association Workshop, pp. 119–123 (2017)
98. Lasko, T.A., Hauser, S.E.: Approximate string matching algorithms for limited-vocabulary ocr output correction. In: Document Recognition and Retrieval VIII, vol. 4307, pp. 232–240. International Society for Optics and Photonics (2000)
99. Melo, C.A.B., Dueire Lins, R.: A comparative study on ocr tools. In: Vision Interface, pp. 1–9 (1999)
100. Mengerink, J.: The dsl/model co-evolution problem in industrial mde ecosystems. Ph.D. thesis, Mathematics and Computer Science (2018)
101. Mohagheghi, P., Dehlen, V.: Where is the proof? - a review of experiences from applying mde in industry. In: Model Driven Architecture – Foundations and Applications, pp. 432–443. Springer Berlin Heidelberg (2008)
102. Moreno, V., Génova, G., Alejandres, M., Fraga, A.: Automatic classification of web images as uml diagrams. In: Spanish Conference on Information Retrieval, pp. 1–8 (2016)
103. Mustafiz, S., Denil, J., Lúcio, L., Vangheluwe, H.: The ftg+ pm framework for multi-paradigm modelling: An automotive case study. In: International Workshop on Multi-paradigm Modeling, pp. 13–18 (2012)
104. Perianez-Pascual, J., Rodriguez-Echeverria, R., Burgueño, L., Cabot, J.: Towards the optical character recognition of dsls. In: International Conference on Software Language Engineering, p. 126–132 (2020)
105. Qamar, A., Paredis, C.J., Wikander, J., During, C.: Dependency modeling and model management in mechatronic design. pp. 1–12 (2012)
106. Rashid, S.: Automatic classification of uml sequence diagrams from images (2019)
107. Reis, A., Paulino, D., Filipe, V., Barroso, J.: Using online artificial vision services to assist the blind - an assessment of microsoft cognitive services and google cloud vision. In: Trends and Advances in Information Systems and Technologies, pp. 174–184. Springer International Publishing, Cham (2018)
108. Romano, J., Kromrey, J.D., Coraggio, J., Skowronek, J.: Appropriate statistics for ordinal level data: Should we really be using t-test and cohen'sd for evaluating group differences on the nsse and other surveys. In: Annual Meeting of the Florida Association of Institutional Research, pp. 1–33 (2006)

109. Stahl, T., Voelter, M., Czarnecki, K.: Model-driven Software Development: Technology, Engineering, Management. John Wiley & Sons, Inc. (2006)
110. Störrle, H., Hebig, R., Knapp, A.: An index for software engineering models. In: PSRC@ MoDELs, pp. 36–40 (2014)
111. Sun, Y., Gray, J., Bulheller, K., von Baillou, N.: A model-driven approach to support engineering changes in industrial robotics software. In: International Conference on Model Driven Engineering Languages and Systems, pp. 368–382. Springer (2012)
112. Törngren, M., Qamar, A., Biehl, M., Loiret, F., El-Khoury, J.: Integrating viewpoints in the development of mechatronic products. Mechatronics **24**(7), 745–762 (2014)
113. Torres, W., van den Brand, M.G.J., Serebrenik, A.: Suitability of optical character recognition (ocr) for multi-domain model management. In: International Conference on Systems Modelling and Management, pp. 149–162. Springer (2020)
114. Tovar-Arriaga, S., Vargas, J.E., Ramos, J.M., Aceves, M.A., Gorrostieta, E., Kalender, W.A.: A fully sensorized cooperative robotic system for surgical interventions. Sensors **12**(7), 9423–9447 (2012)
115. Vijayarani, S., Sakila, A.: Performance comparison of OCR tools. International Journal of UbiComp **6**(3), 19–30 (2015)
116. Wohlin, C., Runeson, P., Höst, M., Ohlsson, M.C., Regnell, B., Wesslén, A.: Experimentation in Software Engineering. Springer Science & Business Media (2012)
117. Zhou, Z.D., Gui, L., Tan, Y.G., Liu, M.Y., Liu, Y., Li, R.Y.: Actualities and development of heavy-duty cnc machine tool thermal error monitoring technology. Chinese Journal of Mechanical Engineering **30**(5), 1262–1281 (2017)
118. Zhu, M., Hahn, A., Wen, Y.Q.: Identification-based controller design using cloud model for course-keeping of ships in waves. Engineering Applications of Artificial Intelligence **75**, 22–35 (2018)

8 Appendix

Model ID	Reference
Paper Model 1	Paper [114] Figure 1
Paper Model 1 (2)	Paper [114] Figure 5
Paper Model 1 (3)	Paper [114] Figure 6
Paper Model 1 (4)	Paper [114] Figure 7
Paper Model 1 (5)	Paper [114] Figure 8
Paper Model 1 (6)	Paper [114] Figure 10
Paper Model 1 (7)	Paper [114] Figure 15
Paper Model 2	Paper [93] Figure 1
Paper Model 2 (2)	Paper [93] Figure 2
Paper Model 2 (3)	Paper [93] Figure 3
Paper Model 2 (4)	Paper [93] Figure 5
Paper Model 2 (5)	Paper [93] Figure 7
Paper Model 3	Paper [71] Figure 1 (a)
Paper Model 3 (2)	Paper [71] Figure 1 (b)
Paper Model 3 (3)	Paper [71] Figure 4 (b)
Paper Model 3 (4)	Paper [71] Figure 4 (a)
Paper Model 3 (5)	Paper [71] Figure 5
Paper Model 4	Paper [118] Figure 1
Paper Model 4 (2)	Paper [118] Figure 2
Paper Model 4 (3)	Paper [118] Figure 3 (a)
Paper Model 4 (4)	Paper [118] Figure 5
Paper Model 4 (5)	Paper [118] Figure 6
Paper Model 5	Paper [83] Figure 1
Paper Model 5 (2)	Paper [83] Figure 2
Paper Model 5 (3)	Paper [83] Figure 3
Paper Model 5 (4)	Paper [83] Figure 5
Paper Model 6	Paper [79] Figure 1
Paper Model 6 (2)	Paper [79] Figure 3
Paper Model 7	Paper [117] Figure 1b
Paper Model 7 (2)	Paper [117] Figure 2
Paper Model 7 (3)	Paper [117] Figure 3
Paper Model 7 (4)	Paper [117] Figure 4
Paper Model 7 (5)	Paper [117] Figure 6
Paper Model 7 (6)	Paper [117] Figure 13
Paper Model 7 (7)	Paper [117] Figure 11
Paper Model 7 (8)	Paper [117] Figure 15
Paper Model 7 (9)	Paper [117] Figure 19
Paper Model 7 (10)	Paper [117] Figure 20

Table 11. Mapping between the models id used on plots and scientific papers models.

Model	Median	
	H	SD
Class Diagram (UML) 11	193366	19376665.5
Class Diagram (UML) 12	23332	1651001
Class Diagram (UML) 13	141163.5	10082654
Class Diagram (UML) 14	24841	5756571
Class Diagram (UML) 15	37521.5	5719871
Class Diagram (UML) 16	100636.5	14121234.5
Class Diagram (UML) 17	208570.5	6315443
Class Diagram (UML) 18	36504	10073049.5
Class Diagram (UML) 19	40886	8817458.5
Class Diagram (UML) 20	251315.5	11392504.5
Sequence Diagram (UML) 6	64325.5	12937410.5
Sequence Diagram (UML) 7	84253.5	16274181.5
Sequence Diagram (UML) 8	60969.5	11035539
Sequence Diagram (UML) 9	46443.5	9054040.5
Sequence Diagram (UML) 10	50057.5	12358265.5
Use Case (UML) 6	106869.5	13843530.5
Use Case (UML) 7	31290.5	7213966
Use Case (UML) 8	93645	12330583
Use Case (UML) 9	47635.5	7098907.5
Use Case (UML) 10	65799.5	8499236

Table 12. Execution time (nanoseconds), where "H" means without shape detection and "SD" means with shape detection.

Model	Median	
	H	SD
Matlab Simulink 10	30055.5	3763110
Matlab Simulink 11	45504.5	4968398.5
Matlab Simulink 12	20052	1750365.5
Matlab Simulink 13	157131.5	6586545
Matlab Simulink 14	203516	11716023.5
Matlab Simulink 15	84918	3933299
Matlab Simulink 16	198800.5	11897143
Matlab Simulink 17	134013	4184728
Matlab Simulink 18	75310.5	3957824
Matlab Simulink 19	37194	4139254

Table 13. Execution time (nanoseconds), where "H" means without shape detection and "SD" means with shape detection.

Model	Median	
	H	SD
Paper [118] Figure 1	197934	9530845.5
Paper [118] Figure 2	47069.5	5196559
Paper [118] Figure 3(a)	137875.5	9030410.5
Paper [118] Figure 5	42283.5	4633711.5
Paper [118] Figure 6	60155.5	6121648.5
Paper [83] Figure 1	44364.5	7231225
Paper [83] Figure 2	46901.5	4473500.5
Paper [83] Figure 3	39033	6084878
Paper [83] Figure 5	36963	5423058.5
Paper [79] Figure 1	190011	13709096
Paper [79] Figure 3	36744	4461332.5
Paper [117] Figure 1b	82236	6958840
Paper [117] Figure 2	158594	5860029.5
Paper [117] Figure 3	334828.5	24443442.5
Paper [117] Figure 4	35532	4463604.5
Paper [117] Figure 6	14578	6429471.5
Paper [117] Figure 13	42872.5	6326256
Paper [117] Figure 11	48810	2534552
Paper [117] Figure 15	23690.5	4500211
Paper [117] Figure 19	58240.5	6104495.5
Paper [117] Figure 20	30315	3486046.5

Table 14. Execution time (nanoseconds), where "H" means without shape detection and "SD" means with shape detection.