Reengenharia e Evolução de um Software para Reconhecimento de Diagramas de Classe UML

Arthur Malmann Becker[1], **João Pablo S. da Silva**[1], **Miguel Ecar**[1]

[1]LabISE - Laboratory of Intelligent Software Engineering
Federal University of Pampa (UNIPAMPA)
Alegrete – RS - Brazil

{arthurbecker.aluno,joaosilva}@unipampa.edu.br, miguel@ecarsm.com

Abstract. *Creating whiteboard sketch diagrams is a common job for professionals working in software development because it allows team members to collaborate. However, there are problems with maintaining documentation and future rework to reconstruct the sketched models. In view of this, a tool was developed that enables the recognition of class diagrams. The objective of our work is to carry out a technological update and remedy some existing limitations. Thus, we identified possible improvements to the developed version, and finally evaluated the tool. This work contributes to the reengineering of the project, which attributed modularity and scalability to the project.*

Resumo. *Criar esboços de diagramas em quadro branco é um trabalho comum para profissionais que trabalham com o desenvolvimento de software, pois permite que os membros da equipe colaborem. Porém, existem problemas com a manutenção da documentação e o retrabalho futuro para reconstruir os modelos esboçados. Em vista disso, foi desenvolvida uma ferramenta que possibilita o reconhecimento de diagramas de classe. O objetivo do nosso trabalho é realizar uma atualização tecnológica e sanar algumas limitações existentes. Assim, nós identificamos as possíveis melhorias da versão desenvolvida, e por fim avaliamos a ferramenta. Este trabalho tem como contribuição a reengenharia do projeto, que atribuiu modularidade e escalabilidade ao projeto.*

1. INTRODUÇÃO

Um modelo de software é uma simplificação da realidade, ou seja, uma abstração do mundo real elaborada com a finalidade de compreender o problema antes de implementar a solução [Booch et al. 2006, Blaha and Rumbaugh 2006]. Os modelos abstratos do projeto de software são construídos durante o processo de engenharia de software, denominado modelagem de sistema. Nesse processo ocorre o estudo das informações para a construção dos modelos, para apresentar uma visão ou perspectiva diferente do sistema [Sommerville 2010].

A modelagem requer alguma linguagem capaz de expressar as abstrações do mundo real. A *Unified Modeling Language* (UML) é uma linguagem de modelagem de propósito geral mantida pela *Object Management Group* (OMG) que serve para especificar, construir e documentar os artefatos dos sistemas. A UML encontra-se na versão 2.5 e abrange 14 diagramas, divididos em diagramas estruturais e diagramas comportamentais. Os diagramas estruturais mostram a estrutura estática dos objetos em um sistema,

enquanto os comportamentais mostram o comportamento dinâmico dos objetos em um sistema [OMG 2015]. Dentre esses diagramas, destacamos o diagrama de classe, um dos mais utilizados na modelagem de sistemas [Fowler 2007].

A criação de esboços[1] de diagramas é uma tarefa frequente entre os times e profissionais de software, sendo comum o uso de quadro branco para tal. Isso se dá pelo fato de que quadros brancos não impõem regras e possibilitam a colaboração simultânea dos membros da equipe [Fowler 2007]. Assim, os esboços exercem um importante papel no trabalho de desenvolvimento e manutenção de software, visto que, apoiam o raciocínio para a resolução de um problema, ajudando na transferência do conhecimento de tácito para explícito [Baltes and Diehl 2014, Cherubini et al. 2007].

Outro uso da prática de criação de esboço é na intenção de contribuir nas atividades iniciais da engenharia de software, incluindo reunir e documentar as necessidades dos *stakeholders*[2] do futuro produto de software [Wüest et al. 2013]. O trabalho realizado por [Baltes and Diehl 2014] evidencia a importância dos esboços e diagramas como um recurso valioso, mostrando também que os profissionais de software estão dispostos a manter tais artefatos visuais.

A modelagem de esboços de diagrama à mão possui alguns benefícios bem definidos, tais como: simplicidade e facilidade de uso. Por outro lado, ela contém problemas em relação à integração dos artefatos de software e a sua documentação em imagens, pois a manutenibilidade requer o gerenciamento de imagens em algum repositório de projeto. O retrabalho é outro problema, pois para gerenciar os requisitos adequadamente, o desenvolvedor terá que reconstruir os modelos esboçados em uma ferramenta *Computer-Aided Software Engineering* (CASE) [Wüest et al. 2013, Cherubini et al. 2007].

Esse processo de recriação, para processamento posterior, consome muito tempo e é sujeito a erros, o que pode levar à perda de informações ou interpretações incorretas [Cherubini et al. 2007, Wüest et al. 2013]. Levando em consideração tais fatores, [Giordano 2015] desenvolveu uma aplicação móvel que possibilita a integração entre esboços feitos em quadros brancos e editores UML, através da captura e reconhecimento da imagem de um diagrama de classe em uma perspectiva conceitual. No entanto, a ferramenta desenvolvida possui algumas limitações, por exemplo, a necessidade de baixar uma ferramenta à parte para que o aplicativo funcione. Além disso, o sistema opera apenas no sistema operacional Android e em ambiente controlado, reconhecendo unicamente classes, associações e multiplicidades.

Dado esse contexto, nosso trabalho tem como objetivo geral a reengenharia da solução apresentada por [Giordano 2015], realizando uma atualização tecnológica e sanando algumas das limitações da versão previamente desenvolvida. Complementarmente, nosso objetivo geral pode ser decomposto em efetuar a atualização do estado da arte sobre reconhecimento de esboços de diagramas UML, realizar reengenharia da arquitetura monolítica para arquitetura de microsserviços, adicionar o reconhecimento de mais elementos UML esboçados à mão, e avaliar a ferramenta através de experimentos empíricos.

[1]Um esboço é um instrumento inerentemente tipado e baseado em grafos para formalizar descrições e especificações que podem substituir o método tradicional de linguagem formal, axiomas e regras de inferência [Wells 1990].

[2]Um *stakeholder* é uma pessoa ou papel, o qual é afetado, de alguma forma, pelo sistema [Sommerville 2010].

A fim de atingir nossos objetivos para este trabalho, realizamos um estudo bibliográfico para investigar o estado da arte referente às ferramentas para reconhecimento de esboços de diagramas UML. Após, realizamos uma análise do projeto desenvolvido por [Giordano 2015], na qual identificamos a oportunidade de realizar a reengenharia da arquitetura monolítica para arquitetura de microsserviços. Ainda, realizamos o levantamento dos elementos do diagrama de classe UML que a ferramenta não reconhece, onde adicionamos o reconhecimento de generalizações. Por fim, planejamos e executamos duas avaliações, a primeira junto com usuários finais para avaliar a aceitação do sistema e a segunda para obter métricas de precisão, revocação, acurácia e medida-f do algoritmo.

Assim, este trabalho tem como principal contribuição a reengenharia do projeto, no qual implantamos uma suíte de serviços, que atribuiu modularidade e escalabilidade crescente ao projeto. Outra contribuição é a evolução do algoritmo de reconhecimento, onde foi desenvolvida uma solução para adicionar o reconhecimento de relações do tipo generalização, bem como, a avaliação do algoritmo reconhecedor e da ferramenta. Além disso, contribuímos com a apresentação do estado da arte a partir de uma revisão sistemática na literatura.

O restante deste documento está organizado como segue. Na Seção 2 realizamos a análise do estado da arte e os trabalhos relacionados com o nosso objetivo. Na Seção 3 relatamos o processo de reengenharia e evolução do software de reconhecimento de diagramas esboçados à mão. Na Seção 4 relatamos os experimentos realizado para avaliar o algoritmo reconhecedor e a ferramenta; Finalmente, na Seção 5 apresentamos as nossas considerações finais e as potenciais evoluções através dos trabalhos futuros.

2. Trabalhos Relacionados

[Deufemia and Risi 2020] apresentam um sistema de reconhecimento de esboço para diagramas de multi-domínio desenhados à mão, dentre os domínios abrangidos está o diagramas de classes UML.

O sistema proposto por [Deufemia and Risi 2020] utilizou o processo de segmentação de áreas de interesse da imagem contendo esboços de diagramas UML. Por meio da técnica de programação dinâmica, sua abordagem divide os traços realizados à mão livre em segmentos de formas primitivas, como segmentos de linha e arcos elípticos.

O sistema de reconhecimento de esboço sugerido por [Deufemia and Risi 2020], uma vez conectado a um editor de esboço, realiza uma interpretação à medida que os esboços são feitos. Para isso, utiliza como abordagem o aproveitamento do formalismo *Sketch Grammars*, com duas finalidades, primeiro para definir de forma hierárquica as formas dos símbolos e a sintaxe abstrata das notações diagramáticas, e segundo para gerar os analisadores LR [3]. Além disso, para lidar com a imperfeição dos símbolos desenhados à mão, a técnica de análise sugerida combina técnicas de recuperação de erro estabelecidas para compiladores de linguagem de programação.

Com objetivo de avaliar o desempenho de reconhecimento o autor realizou um experimento, no qual foram recrutados vinte indivíduos com pouca experiência no uso de interfaces baseadas em esboço e com conhecimento básico de diagramas UML. Os resul-

[3]O L no termo LR denota que a entrada é processada da esquerda para a direita, enquanto o R denota que uma derivação à direita é produzida [Venske et al. 2007].

tados foram medidos usando as métricas precisão e recall, onde se obteve indicadores que os símbolos com maior taxa de reconhecimento são os símbolos de Classes, que de 455 ocorrências apenas 3 foram reconhecidos de forma errada, e os símbolos de Pacotes que 78 dos 86 símbolos foram reconhecidos corretamente. Por fim, os resultados fornecem as taxas de reconhecimento dos algoritmos de linha de base (BL) e multicamadas (ML), onde BL reconhece corretamente 79% dos símbolos em média, enquanto ML atinge 90% dos símbolos corretamente identificados.

O software apresentado por [Chen et al. 2008] denominado SUMLOW, permite que diagramas UML sejam esboçados e reconhecidos em um quadro branco eletrônico baseado em caneta e, em seguida, exportados para uma ferramenta CASE convencional.

A ferramenta SUMLOW [Chen et al. 2008], realiza o reconhecimento e formalização progressiva dos elementos do diagrama. No intuito de realizar o reconhecimento das formas é utilizado o reconhecimento de múltiplos gestos, já para os textos é aplicado o reconhecimento de gesto único através do algoritmo de Rubine. Além disso, é aplicado um segundo nível de filtragem com extensões do algoritmo de Apte, nessa etapa são reconhecidos traços dentro de formas geométricas, bem como é realizada uma comparação de dimensionamento, assim permitindo o reconhecimento de símbolos notacionais UML mais complexos. Desse modo, cria-se um novo elemento de esboço em cada canetada realizada pelo usuário, essas formas são reconhecidas como tipos de elementos UML pelo algoritmo de reconhecimento de gestos de múltiplos traços, em um processo de segundo plano. Após o elemento ser reconhecido, ele serve como apoio para auxiliar no reconhecimento de subelementos, ou seja, o que está dentro de sua borda e relacionamentos.

A fim de identificar a precisão do reconhecimento do SUMLOW, o autor realizou uma avaliação que obteve como resultado a taxa de reconhecimento dos elementos, onde dois itens se destacaram com as maiores taxas sendo eles ícones de ativação com 91,7% e de objeto com 90,1%. Além disso, foi possível verificar as piores taxas de reconhecimento, como o ícone do nó com 69,2%, e o elemento de elipse simples que representa um caso de uso com 73,2%.

Para o reconhecimento de esboço em diferentes domínios [Blagojevic et al. 2011] utilizam técnicas de mineração de dados para construir divisores de texto e formas, além disso se faz uso da mineração de dados nos algoritmos de aprendizado de máquina para pesquisar padrões de dados. Esses divisores treinados contêm todas as informações necessárias para um reconhecedor classificar um dado traço. Assim sendo, os esboços são coletados e rotulados, e logo após convertidos em um conjunto de dados para a realização de uma análise de dados subsequente. Então, a mineração de dados usa os algoritmos de aprendizado de máquina para pesquisar padrões nos dados, procurando identificar se o traço pertence a uma forma ou a um texto.

A avaliação realizada pelos autores mostra que a maior área de fraqueza de seus classificadores é em relação ao elemento seta, onde as taxas de reconhecimento variam de 40% a 64%, além disso, os resultados para as linhas também são baixos, variando de 72% a 92% para o conjunto de dados de diagramas ER/Processo. Também verificou-se que o recurso simples de largura da caixa delimitadora pode classificar corretamente cerca de 85% do conjunto de dados de treinamento, e com a adição de sete outros recursos em uma árvore de decisão, o divisor foi capaz de classificar aproximadamente 79% dos

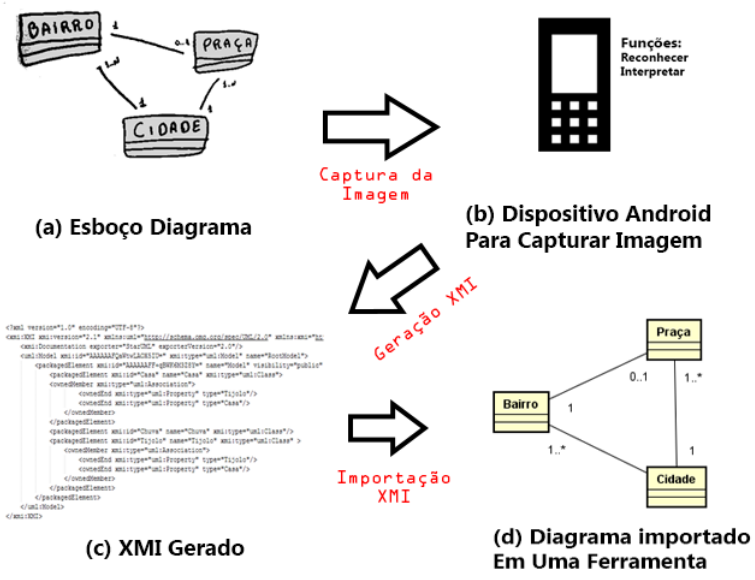
dados. Já o recurso entropia foi capaz de classificar corretamente aproximadamente 83% dos dados. Por fim, uma validação cruzada mostra que 85,76% dos dados de treinamento estão classificados corretamente.

Diante dos dados que analisamos, podemos dizer que todos os algoritmos possuem uma boa taxa de reconhecimento dos elementos de diagramas UML, mesmo nos casos em que alguns elementos tiveram uma menor taxa de reconhecimento. Desse modo, destacamos entre as abordagens apresentadas o algoritmo de aprendizado de máquina e mineração de dados, pois esse algoritmo é capaz de reconhecer um bom percentual dos elementos UML, bem como elementos de outros domínios.

3. Reengenharia e Evolução

[Giordano 2015] desenvolveu um aplicativo que, conforme a estrutura apresentada na Figura 1, captura a imagem utilizando um dispositivo móvel Android. O sistema reconhece o diagrama de classe UML a partir da imagem, e logo após gera um arquivo *XML Metadata Interchange* (XMI) para ser importado em um editor UML. Assim, resolvendo a questão do retrabalho de integrar o esboço de um diagrama com uma ferramenta CASE, desse modo minimizando o trabalho das equipes.

Figura 1. Estrutura da solução do autor [Giordano 2015].



A ferramenta desenvolvida é capaz de efetuar a detecção e reconhecimento de classes e seus respectivos nomes, multiplicidades e relacionamentos do tipo associação simples não direcional. O reconhecimento dos elementos mencionados ocorre apenas em um ambiente controlado, o qual deve seguir as seguintes regras:

- as associações não devem encostar nas classes;
- as multiplicidades não devem encostar nas relações;
- as classes devem estar fechadas (aberturas mínimas são retiradas por meio da morfologia matemática, porém aberturas maiores não);
- o nome da classe deve ser escrito em letra de forma o mais legível possível;

- o nome da classe não deve encostar nas bordas da classe.

Em virtude dos aspectos abordados, iniciamos nosso trabalho com a realização de uma análise, na qual identificamos as oportunidades de reengenharia da arquitetura monolítica para arquitetura de microsserviços. Com o projeto já na nova arquitetura, efetuamos uma segunda análise, agora com o objetivo de identificar elementos de um diagrama de classe que a ferramenta não reconhece atualmente. Por fim, planejamos e implementamos uma abordagem para aprimorar o algoritmo, adicionando o reconhecimento de mais um elemento UML.

3.1. Aprimoramento do Algoritmo de Reconhecimento

A segunda etapa de desenvolvimento consiste na aprimoramento do algoritmo de reconhecimento. Assim, efetuamos uma análise para identificar os principais elementos de um diagrama de classe que a ferramenta não reconhece. Na versão atual a ferramenta suporta o reconhecimento de classes, que ocorre em duas etapas, primeiro extraímos da imagem os elementos que possuírem o tamanho da área maior que 1000 pixels e também possuir um nome. Logo após, interpretamos o nome da classe através do OCR chamado de Tesseract, no momento treinamos esse OCR com o arquivo de dados do idioma português para letras de forma. Para realizar o treinamento, foi necessário apenas passar por parâmetro o caminho do arquivo de treinamento [4].

Nosso programa também permite o reconhecimento de associações simples, as quais são distinguidas dos outros elementos devido seu contraste de largura e a altura, ou seja, consideramos o elemento uma relação se a altura for cinco vezes maior que a largura ou vice-versa. Em seguida verificamos as classes que cada relação pertence, para isso, identificamos as classes que estão conectadas por meio da distância entre o centro da classe e a borda da associação. Fazemos o cálculo da medida de distância mediante a fórmula derivada do Teorema de Pitágoras, conforme exibimos na Equação 1.

$$\sqrt{distancia = (pBX - pAX)^2 + (pBY - pAY)^2} \tag{1}$$

O tamanho do esboço da classe e a resolução do dispositivo costumam ser bastante variáveis em nosso cenário, então é a melhor solução usarmos valores fixos para verificar se a classe está conectada ou não. Dessa forma, utilizamos o tamanho médio da classe, assim calculamos a distância permitida usando a Equação 2, que leva em consideração a altura e a largura ao determinar a distância de conexão permitida entre os elementos. Também considerando um determinado valor, para contornar o fato das relações não estarem diretamente ligadas nas classes.

$$distanciaAceitavel = altura + largura/1.8 \tag{2}$$

Reconhecemos os elementos como uma multiplicidade quando sua área for maior que 100 pixels e menor que 1000 pixels. Nesse caso, assim como nas relações utilizamos o cálculo de proximidade, com a diferença de que as multiplicidades estão mais próximas das conexões neste cenário. Por fim, reconhecemos os caracteres com o OCR.

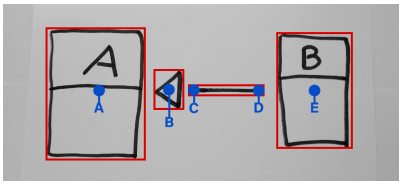
[4]Os arquivos para treinamento do Tesseract OCR podem ser obtidos no repositório do Tesseract no GitHub: `https://github.com/tesseract-ocr/tessdata_fast`

Diante dos fatos analisados, identificamos que no momento a ferramenta não consegue reconhecer relações do tipo: dependência, generalização, agregação e composição. Desse modo, codificamos uma solução para reconhecer as relações de generalização.

O reconhecimento de heranças consistem em extrair os elementos com área maior que 100 pixels e menor que 1000 pixels. Feito isso, identificamos o número de vértices de cada um dos elementos através do método *ApproxPolyDP* do Open CV. Caso o elemento possua 3 vértices é classificado como um triângulo, e definimos que a relação é uma generalização. Por fim é realizado um cálculo de proximidade com dois objetivos, primeiro para identificarmos em qual extremidade da relação o símbolo se encontra e depois para verificarmos com qual classe ele está sendo relacionado.

Assim, primeiro é realizada a extração dos elementos, após, na etapa de classificação localizamos os elementos com 3 vértices e definimos como um triângulo, em seguida verificamos os elementos com a altura cinco vezes maior que a largura ou vice versa e identificamos como uma relação. Já os elementos são classificados como uma classe, quando o mesmo possuir o tamanho da área maior que 1000 pixels e também possuir um nome. Por fim, definimos os elementos com área maior que 100 pixels e menor que 1000 pixels como uma multiplicidade. Na Figura 2 exemplificamos esse processo, onde os retângulos vermelhos identificam os elementos detectados e extraídos. Os pontos “A”, “B” e “E” simbolizam os centros dos elementos e os pontos “C” e “D” as extremidades da relação.

Figura 2. Reconhecimento de Generalizações.



Com os elementos já extraídos, localizamos e pegamos o ponto central do triângulo (B), e calculamos a proximidade desse ponto com o centro de cada classe existente (A e E), por fim pegamos a classe com o centro mais próximo (A) e a identificamos como *General*. A partir do centro do triângulo (B) calculamos qual a relação com extremidade mais próxima (C), com isso pegamos a outra extremidade (D) e calculamos qual centro de classe está mais próximo (E), e identificamos a classe como *Specific*. Por fim, com os elementos do modelo devidamente classificados e com suas conexões mapeadas, realizamos a geração do XMI.

4. Avaliação

Nesta seção expomos como foi realizado o processo das avaliações realizadas. Primeiro avaliamos o algoritmo de reconhecimento, onde extraímos as métricas de precisão, revocação, acurácia e medida-f do algoritmo. Por fim, também realizamos a avaliação junto com usuários finais do sistema, para conferir a aceitação da ferramenta em um contexto real de desenvolvimento.

4.1. Avaliação do Algoritmo Reconhecedor

Com o propósito de avaliar o algoritmo de classificação e reconhecimento, realizamos a criação e captura de 25 diagramas de classe com seus respectivos XMI, para capturar

a imagem foi utilizado o smartphone Galaxy S20FE. Cada diagrama foi esboçado com elementos que deveriam ser reconhecidos e elementos que não deveriam ser reconhecidos. Também vale ressaltar que os diagramas foram esboçados seguindo o seguinte ambiente controlado:

- as associações não devem encostar nas classes;
- as multiplicidades não devem encostar nas relações;
- as heranças não devem encostar nas classes;
- na herança o triângulo não deve encostar na linha;
- as classes devem estar fechadas;
- o nome da classe deve ser escrito em letra de forma o mais legível possível;
- o nome da classe não deve encostar nas bordas da classe.

Assim, seguindo o método de avaliação apresentado por [Tharwat 2020], criamos uma matriz de confusão para cada tipo de elemento reconhecido pela ferramenta, como apresentado na Tabela 1. A matriz indica os erros e acertos da solução, comparando com o resultado esperado. Desse modo, na matriz foram informados os seguintes valores:

- **Verdadeiros Positivos (VP):** quando a solução reconhece que é um determinado elemento e, ao verificar o diagrama, vê-se que é realmente o elemento que estava desenhado;
- **Verdadeiros Negativos (VN):** quando a solução não deve reconhecer que é um determinado elemento e, ao verificar o diagrama, vê-se que realmente o elemento não estava desenhado;
- **Falsos Positivos (FP):** quando a solução reconhece que é um determinado elemento e, ao verificar o diagrama, vê-se que o elemento não estava desenhado.
- **Falso Negativo (FN):** quando a solução não reconhece um determinado elemento e, ao verificar o diagrama, vê-se que o elemento estava desenhado.

Tabela 1. Matriz de Confusão de Cada Elemento.

Elemento	VP	VN	FP	FN
Classe	46	4	8	0
Nome da Classe	28	4	0	12
Herança	20	4	2	0
Associação	24	8	10	2
Multiplicidade	20	8	0	12

Com o levantamento de todas as informações da matriz de confusão de cada tipo de elemento que a ferramenta abrange, foi possível obter como resultados as seguintes métricas:

- **Precisão:** proporção do reconhecimento previsto corretamente para o total de reconhecimentos, ou seja, a capacidade de evitar os falsos positivos;
- **Revocação:** é a proporção de reconhecimentos corretos e o total de reconhecimentos realizados, indicando o quão bom foi o reconhecedor na hora de classificar o elemento;
- **Acurácia:** indica uma performance geral do reconhecedor, onde dentre todas as classificações, quantas o reconhecedor classificou corretamente;
- **Medida-F:** é a média harmônica entre precisão e revocação, podendo ser interpretada como uma medida de confiabilidade da acurácia.

Na Tabela 2 é possível visualizar os resultados. No qual observamos que a solução apresentou bons resultados, onde a classificação dos elementos teve uma precisão alta, mostrando a capacidade da solução de evitar falsos positivos. Além disso, vale destacar a acurácia obtida nas Classes e Heranças que está mais alta em relação aos demais elementos que possuem vínculo com o OCR, o que leva a considerar que é necessário treinar mais o reconhecedor de caracteres.

Tabela 2. Resultados da Avaliação do Algoritmo.

Elemento	Precisão	Revocação	Acurácia	Medida F
Classe	$\approx 85\%$	92%	$\approx 86\%$	$\approx 88\%$
Nome da Classe	100%	87%	$\approx 72\%$	$\approx 93\%$
Herança	$\approx 90\%$	$\approx 83\%$	$\approx 92\%$	$\approx 87\%$
Associação	$\approx 70\%$	75%	$\approx 72\%$	$\approx 72\%$
Multiplicidade	100%	$\approx 71\%$	70%	$\approx 83\%$

No caso das multiplicidades, a baixa acurácia pode se dar devido ao fato de a mesma conter um ou dois caracteres, o que impede do OCR Tesseract inferir uma palavra no caso de reconhecimento parcial. Já no caso dos nomes de classe, a acurácia pode ser afetada pela grande variação de caligrafia com diferentes tipos de letras.

Dentre os elementos, destaca-se a associação com o pior resultado. Durante a execução dos testes, foi possível observar que o reconhecimento da associação acaba sendo impactada pelo reconhecimento da multiplicidade. Onde em alguns casos, o erro no reconhecimento de uma multiplicidade provoca uma classificação e reconhecimento errado de uma associação.

Por fim, vale destacar que durante a execução da avaliação foi possível encontrar alguns problemas de exceções na ferramenta, onde 2 diagramas não conseguiram ser reconhecidos, devido a imagem conter uma grande quantidade de ruídos, por exemplo, sombras e reflexos. Assim, sendo necessário criar na etapa de pré-processamento, um tratamento de exceção para casos que a ferramenta não consegue reconhecer o esboço.

5. Considerações Finais

O principal objetivo do nosso trabalho é a evolução da solução apresentada por [Giordano 2015], realizando uma atualização tecnológica e sanando parte das suas limitações. Para atingirmos esse objetivo, efetuamos uma atualização do estado da arte sobre reconhecimento de esboços de diagramas UML. Por fim, realizamos a avaliação da ferramenta e do algoritmo de reconhecimento.

Nossa principal contribuição é a reengenharia do projeto, partir da qual implantamos uma suíte de serviços, que atribuiu modularidade e escalabilidade crescente ao projeto, possibilitando que o sistema possa operar em navegadores e nos sistemas operacionais Android e iOS.

Salienta-se que, com a avaliação do algoritmo reconhecedor podemos afirmar que a solução apresentou uma alta precisão e acurácia na classificação e reconhecimento dos elementos dentro do ambiente controlado.

Como trabalhos futuros para essa pesquisa, destacamos a possibilidade de realizar a implementação de uma solução para adicionar o reconhecimento de mais elementos do diagrama de classes, como por exemplo atributos e métodos.

Referências

Baltes, S. and Diehl, S. (2014). Sketches and diagrams in practice. In *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, FSE 2014, page 530–541, New York, NY, USA. Association for Computing Machinery.

Blagojevic, R., Plimmer, B., Grundy, J., and Wang, Y. (2011). Using data mining for digital ink recognition: Dividing text and shapes in sketched diagrams. *Computers Graphics*, 35(5):976–991.

Blaha, M. and Rumbaugh, J. (2006). *Modelagem e projetos baseados em objetos com UML 2*. Elsevier.

Booch, G., Rumbaugh, J., and Jacobson, I. (2006). *UML: guia do usuário*. Elsevier Brasil.

Chen, Q., Grundy, J., and Hosking, J. (2008). Sumlow: Early design-stage sketching of uml diagrams on an e-whiteboard. *Software - Practice and Experience*, 38(9):961–994. cited By 30.

Cherubini, M., Venolia, G., DeLine, R., and Ko, A. J. (2007). Let's go to the whiteboard: How and why software developers use drawings. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '07, page 557–566, New York, NY, USA. Association for Computing Machinery.

Deufemia, V. and Risi, M. (2020). Multi-domain recognition of hand-drawn diagrams using hierarchical parsing. *Multimodal Technologies and Interaction*, 4(3):1–30. cited By 0.

Fowler, M. (2007). *UML Essencial*. Bookman editora, Porto Alegre, RS, Brasil, 3 edition.

Giordano, D. M. (2015). Uml sketch recognizer: um aplicativo para reconhecimento de esboços de diagramas de classe em fotos.

OMG (2015). Omg unified modeling language.

Sommerville, I. (2010). *Software Engineering*. Addison-Wesley Publishing Company, Boston, MA, USA, 9 edition.

Tharwat, A. (2020). Classification assessment methods. *Applied Computing and Informatics*.

Venske, S. M. G. S., Boss, S. L. B., Musicante, M. A., Soares, I. W., Agner, L. T. W., de Ré, A. M., and Dall'Agnol, J. M. H. (2007). Uma extensão do algoritmo de parser lr (0). *Publicatio UEPG: Ciências Exatas e da Terra, Agrárias e Engenharias*, 13(01).

Wells, C. (1990). A generalization of the concept of sketch. *Theoretical Computer Science*, 70(1):159–178.

Wüest, D., Seyff, N., and Glinz, M. (2013). Flexisketch: A mobile sketching tool for software modeling. In Uhler, D., Mehta, K., and Wong, J. L., editors, *Mobile Computing, Applications, and Services*, pages 225–244, Berlin, Heidelberg. Springer Berlin Heidelberg.