UNIVERSIDADE FEDERAL DE SANTA MARIA
CENTRO DE TECNOLOGIA
CURSO DE CIÊNCIA DA COMPUTAÇÃO

AUTOMAÇÃO DE INTERFACES GRÁFICAS PARA MODELOS DE SIMULAÇÃO DE CULTURAS AGRÍCOLAS COM BASE EM LINGUAGEM DE PROGRAMAÇÃO VISUAL

TRABALHO DE GRADUAÇÃO

Romulo Pulcinelli Benedetti

Santa Maria, RS, Brasil

2016

AUTOMAÇÃO DE INTERFACES GRÁFICAS PARA MODELOS DE SIMULAÇÃO DE CULTURAS AGRÍCOLAS COM BASE EM LINGUAGEM DE PROGRAMAÇÃO VISUAL

Romulo Pulcinelli Benedetti

Trabalho de Graduação apresentado ao Curso de Ciência da Computação da Universidade Federal de Santa Maria (UFSM, RS), como requisito parcial para a obtenção do grau de

Bacharel em Ciência da Computação

Orientadora: Profł. Drł. Andrea Schwertner Charão

412
Santa Maria, RS, Brasil

2016

Universidade Federal de Santa Maria
Centro de Tecnologia
Curso de Ciência da Computação

A Comissão Examinadora, abaixo assinada,
aprova o Trabalho de Graduação

AUTOMAÇÃO DE INTERFACES GRÁFICAS PARA MODELOS DE SIMULAÇÃO DE CULTURAS AGRÍCOLAS COM BASE EM LINGUAGEM DE PROGRAMAÇÃO VISUAL

elaborado por
Romulo Pulcinelli Benedetti

como requisito parcial para obtenção do grau de
Bacharel em Ciência da Computação

COMISSÃO EXAMINADORA:

Andrea Schwertner Charão, Drl.
(Presidente/Orientadora)

Nereu Augusto Streck, Prof. Dr. (UFSM)

João Vicente Ferreira Lima, Prof. Dr. (UFSM)

Santa Maria, 06 de Julho de 2016.

RESUMO

Trabalho de Graduação
Curso de Ciência da Computação
Universidade Federal de Santa Maria

AUTOMAÇÃO DE INTERFACES GRÁFICAS PARA MODELOS DE SIMULAÇÃO DE CULTURAS AGRÍCOLAS COM BASE EM LINGUAGEM DE PROGRAMAÇÃO VISUAL

AUTOR: ROMULO PULCINELLI BENEDETTI
ORIENTADORA: ANDREA SCHWERTNER CHARÃO
Local da Defesa e Data: Santa Maria, 06 de Julho de 2016.

A automação de tarefas é uma forma bastante eficiente pela qual podemos reduzir custos e aumentar a produtividade e qualidade da atividade humana. A computação por si é uma ferramenta para atingir a automação de tarefas, com vários exemplos de softwares focados em automatizar tarefas específicas sob comando do usuário, sendo a automatização destes softwares um campo a parte. Observamos a aproximação destes softwares a abordagens mais naturais ao raciocínio humano, por meio de interfaces gráficas e contextualização dos elementos com base no mundo real, tornando estes softwares menos distantes do paradigma de interação do humano com a realidade. Desta forma este trabalho objetiva abordar a utilização de tecnologias voltadas a programação visual, em especial a biblioteca para criação de linguagens visuais, Blockly, para melhorar a abordagem de automação de tarefas, com foco em softwares gráficos de modelagem matemática agrícola, assim inserindo a atividade de automatizar estes softwares gráficos, dentro do mesmo domínio de abstração que as atividades destes softwares ocorrem.

Palavras-chave: Automação de Software. Linguagens de Programação Visual. Modelos de Simulação de Culturas Agrícolas. Automação de Interfaces Gráficas. Blockly.

SUMÁRIO

1 INTRODUÇÃO

No desenvolvimento de software, áreas de conhecimento como engenharia de software e qualidade de software investigam processos e normas, com o objetivo de reduzir a quantia de recursos necessários e garantir a qualidade de software produzido. Um dos produtos destas áreas, envolvendo automação, foi o campo de conhecimento de testes de software.

Considerando que o software pode realizar diversas tarefas, e estas podem assumir diversos estados, qualquer abordagem de teste de software, numa situação ideal deveria avaliar todas estas tarefas e seus estados para fornecer as melhores informações possíveis sobre a qualidade do software, o que facilmente pode se tornar uma tarefa repetitiva e de longa duração e na maioria dos casos nem sempre é possível, segundo (MYERS; SANDLER; BADGETT, 2011, pag. 10).

Embora nem sempre seja possível testar todos os estados possíveis, é possível realizar os testes em uma faixa conhecida e finita de estados. Nestes casos, usar ferramentas como *frameworks* voltados a automatização destes testes agiliza a tarefa de repetir os testes, como na abordagem de testes de regressão onde todos os testes devem ser executados novamente a cada ciclo de desenvolvimento.

Estas ferramentas oferecem também uma série de vantagens tais como precisão ao reduzir a necessidade da atenção humana durante o andamento da tarefa, acelerando atividades e melhorando o aproveitamento do tempo de trabalho humano.

Alguns destes *frameworks*, embora voltados a realização de testes, poderiam ser usados para automatizar programas gráficos. Um exemplo disso é o Robot Framework (ROBOT FRAMEWORK, 2016). Entretanto, são ferramentas que, apesar de cobrirem de forma bastante detalhada a automação de testes, não são as mais adequadas à modelagem da automatização de softwares gráficos, dependem de um domínio de assuntos de diversos campos da área de Ciências da computação e em muitos casos, domínio da especificação e arquitetura do software a ser automatizado ou ainda alterações a nível de codificação no programa a ser automatizado.

Já outros casos particulares de automatização focada em tarefas de TI são os próprios terminais ou ainda utilitários como *shell*, *make* e afins. Tratam-se de ferramentas e linguagens voltadas para automatização do desenvolvimento e de tarefas administrativas, também inadequadas à automatização de programas gráficos, seja por serem bastante limitadas a um mundo formalmente textual, pela complexidade de sua sintaxe ou ainda pela abstração focada em ta-

refas comuns apenas para profissionais de TI e para software que oferece interface com estes utilitários.

Existem também ferramentas destinadas à automação de software gráfico, como a linguagem *script* de automatização Autoit (AUTOIT, 2015). Ainda assim, é uma ferramenta que exige o domínio de um nível de abstração elevado, perceptivelmente diferente da abstração em que a atividade se dá, em um ambiente gráfico, focado em facilidades visuais de interação.

A automatização de tarefas pode ser aproximada do usuário por meio de abordagens mais visuais e sintaxe mais contextualizada à modelagem de tarefas genéricas em interfaces gráficas, com o uso de linguagem visual para descrição das automações, onde um exemplo de ferramenta para construção de linguagens visuais é a biblioteca Blockly (BLOCKLYRESOURCE, 2016) usada neste projeto.

Um tipo de software que se beneficiaria da automatização de tarefas é o de simulação de culturas agrícolas (do inglês *Crop Models*). Os modelos de simulação vem sendo refinados para prever o comportamento (por exemplo, taxa de crescimento) de diferentes cultivares sob determinadas condições do ambiente (por exemplo, volume de chuvas). Alguns exemplos de modelos em uso hoje em dia são SoySim (SOYSIM, 2016) (soja), Simanihot (SIMANIHOT, 2016) (mandioca) e DSSAT (*Decision Support System for Agrotechnology Transfer*) (DECISION SUPPORT SYSTEM FOR AGROTECHNOLOGY TRANSFER , DSSAT) (diversas culturas). Embora estejam se tornando significativamente mais fáceis de interagir por via gráfica, dependendo das atividades realizadas e da forma como o modelo as realiza, trabalhar com estes programas pode se tornar uma atividade manual repetitiva e demorada. Tarefas como simulações em climas futuros é um dos exemplos mais notórios desta situação.

A automatização destes modelos via uma abordagem mais visual permitiria a obtenção das vantagens aqui discutidas, sem deslocar o utilizador de sua área de domínio, a interface gráfica.

1.1 Objetivos

1.1.1 Objetivo Geral

O principal objetivo deste trabalho é tornar possível a automação de tarefas em interfaces gráficas por meio da simplificação de uma abordagem visual com base na biblioteca de programação visual Blockly (BLOCKLYRESOURCE, 2016), dentro do contexto de modelos

de simulação de culturas agrícolas.

1.1.2 Objetivos Específicos

- Fornecer uma solução menos formal e textual, mais visual e contextualizada, de automatização;
- Automatizar tarefas computacionais em interfaces gráficas no campo de modelagem matemática agrícola;
- Auxiliar o campo de pesquisa e trabalho com modelos matemáticos de culturas agrícolas;

1.2 Justificativa

A automação de tarefas é hoje em dia um processo fundamental para a obtenção de resultados ágeis e de qualidade, tanto na produção de um produto ou no fornecimento de serviços, assim como na execução de tarefas pessoais. Representa também redução de custos, o que abre novas possibilidades permitindo trabalhos mais complexos e maiores chances de sucesso. Entretanto, ferramentas de automação na computação têm exigido um domínio de abordagens de abstração que, em geral, vão além da abstração com a qual, usuários finais de outras áreas estão acostumados. Uma abstração mais próxima ao nível em que estas tarefas ocorrem tornaria a automação um processo mais intuitivo.

Uma destas áreas de conhecimento é a Fitotecnia, mais especificamente, estudos do desenvolvimento do vegetal e produtividade de culturas que hoje utilizam modelos de simulação de culturas agrícolas. Essa área evoluiu para programas com interfaces gráficas, objetivando alcançarem e beneficiarem mais pessoas, leigas em computação porém fluentes na área de conhecimento da ferramenta. Em alguns casos, as tarefas realizadas com estes simuladores envolvem interações repetitivas para obter uma grande faixa de amostragem de resultados, tarefas estas que se beneficiariam da automatização e estariam igualmente acessíveis ao domínio de seus usuários, caso esta automatização pudesse ser realizada dentro deste domínio de entendimento, um ambiente e uma metodologia visual.

2 REVISÃO DE LITERATURA

Na sequência serão apresentados conceitos relativos aos conteúdos abordados nesse trabalho, descrevendo a automação, sua utilização dentro computação na área de TI e os resultados até então obtidos na automatização de tarefas mais genéricas, assim como linguagens de programação visual e o Blockly, ferramenta com a qual este aspecto será tratado.

2.1 Automação e suas abordagens dentro da computação

A automação é a execução de tarefas por meio de máquinas e computadores, antes executáveis apenas por humanos (PARASURAMAN; SHERIDAN; WICKENS, 2000). Segundo NOF (NOF, 2009, pág. 124), a automação teve um impacto significativo na economia e desenvolvimento tecnológico da sociedade. É um elemento-chave para o alcance de produtos e serviços de alta qualidade e baixo custo. Uma das áreas impactadas pela automação foi a precisão em um ciclo autoalimentado, onde a automação melhora a precisão e por sua vez a precisão melhora a automação (DONMEZ; SOONS, 2008).

A automação da informação por meio de computadores, segundo NOF (NOF, 2009, pág. 3) é um processo dos dias atuais, onde temos vendas automatizadas de passagens, conexões de chamadas em nossos telefones, realizadas de forma automática, dentre outras mudanças advindas da automatização. Temos também a produção e manufatura por intermédio da robótica. No final das contas, o impacto da informática promoveu não só uma intensa automatização passiva, mas também, segundo VENKATRAMAN(VENKATRAMAN, 1994), tem criado e mantido flexíveis redes de negócios, inclusive transformando a forma como realizamos negócios e atividades.

Dentro da área de computação, observamos a automação agilizar e refinar tarefas como testes, desenvolvimento e manutenção de projetos, devido ao reconhecimento de que o desenvolvimento de software muitas vezes consiste na criação sistemática de componentes que devem aderir a um conjunto bem específico de restrições (BARRY; KEMERER; SLAUGHTER, 2007).

Segundo BARRY; KEMERER; SLAUGHTER(BARRY; KEMERER; SLAUGHTER, 2007), a automação do processo de desenvolvimento tem o potencial de reduzir o erro humano em código que deve se adequar a sintaxe e restrições precisas, podendo inclusive produzir software de melhor qualidade que o produzido manualmente, considerando que o talento em desenvolvimento de software é escasso, representando também uma redução de custo. Também reduz

a necessidade de interação humana com tarefas secundárias ou de pouco interesse, contribuindo para redução da complexidade da tarefa.

Dentre tarefas comuns na área de TI, temos a automatização de tarefas administrativas via linguagens e ferramentas tais como um *shell* e linguagem *script* específica, ou ainda ferramentas de automatização focadas em tarefas de desenvolvimento, como *make*, ferramenta de automação de *builds*, ou como Robot Framework, ferramenta voltada à automação de testes (UNIX SHELL, 2015; GNU MAKE, 2015; ROBOT FRAMEWORK, 2016).

Considerando que TI não é a única área onde tarefas repetitivas, bem definidas e restritas ocorrem, o interesse em automatizar estas tarefas resultou em software tal como o AutoIt(AUTOIT, 2015), para plataforma Windows, um programa que permite automatizar programas com interface por meio da descrição da automatização em uma linguagem *script*, podendo gerar executáveis independentes que rodam em computadores que não tenham o AutoIt instalado. Essa ferramenta tem à disposição uma grande diversidade de bibliotecas com funções prontas, tendo inclusive uma IDE voltada para sua linguagem de automação. Outro exemplo é o Automator, que permite automatizar tarefas repetitivas em plataformas Macintosh, permitindo construir *workflows* por meio de unidades modulares chamadas ações; apesar de conter diversas ações pre estabelecidas, é possível inserir novas ações por meio de linguagens como AppleScript e Objective-C (AUTOMATOR, 2007).

Existem outras ferramentas que se propõem a solucionar estes problemas de automatização em interfaces gráficas, com características diversas, algumas delas proprietárias, outras com uma linguagem mais visual e informal que linguagens *script* e código tradicional, tais como o UiPath, Sekulix e o TestComplete (UIPATH, 2016; TESTCOMPLETE, 2016; SIKULI, 2014), ferramentas que abordam a automação de uma forma mais abstrata, porém com uma flexibilidade lógica limitada por esta abstração.

2.2 Programação visual

Segundo SHU(SHU, 1988), programação visual é "o uso de representações gráficas significativas no processo de programação" realizada em uma linguagem que SHU define como "uma linguagem que utiliza alguma representação visual para completar o que outrora deveria ser escrito em uma linguagem unidimensional tradicional". Esta definição hoje tem se mostrado um tanto ampla e tem apresentado diversas contextualizações como podemos ver em (KOEGEL; HEINES, 1993), que apresenta programação visual no contexto de autoria multimídia.

No contexto do presente projeto, programação visual representa a programação de tarefas de automatização de programas com interface gráfica, com base em elementos visuais por meio do Blockly (BLOCKLYRESOURCE, 2016). Esta é uma biblioteca de código aberto destinada à criação de editores para programação visual, totalmente baseada em tecnologias web e portável. Trata-se de uma ferramenta que executa do lado do cliente, funcionando na maioria dos navegadores web e em dispositivos móveis (BLOCKLYRESOURCE, 2016).

Blockly pode ser integrado a qualquer aplicação que ofereça um componente web e é capaz de oferecer teste de unidade, possibilidade de tradução, tratamento de eventos e construção de blocos customizáveis, permitindo inclusive mutabilidade. O processo de criação dos blocos consiste em definir seu formato, campos e pontos de conexão e se necessário, modificadores, o que pode ser realizado com o uso do Block Factory ou ainda da API JSON. Em seguida é criado o gerador de código para que o novo bloco possa ser exportado para alguma linguagem de programação.

Um exemplo típico de definição de um bloco:

```
Blockly.Blocks['text_length'] = {
    init: function() {
        this.setColour(160);
        this.appendValueInput('VALUE')
            .setCheck('String')
            .appendField('length of');
        this.setOutput(true, 'Number');
        this.setTooltip('Returns number of
            letters in the provided text.');
        this.setHelpUrl('http://www.w3schools.com/
            jsref/jsref_length_string.asp');
    }
};
```

Embora Blockly não seja escalável para grandes programas, pode ainda ser usado como um editor de linguagens visuais para áreas específicas, apresentando elementos comuns a linguagens de programação tais como funções, variáveis, *arrays*, checagem básica de tipos e afins. Por ser uma ferramenta para criação de editores de linguagens visual, impossibilita que o usuário cometa erros de sintaxe ao fazer programas via Blockly.

A flexibilidade da ferramenta pode ser observada na utilização para criação de jogos, aplicativos móveis para Android, programação web ou ainda como recurso educacional (BLOCKLYGAMES, 2016; APPINVENTOR, 2015; MARRON; WEISS; WIENER, 2012; SILVEIRA JúNIOR et al., 2015).

2.3 Modelos matemáticos de simulação de culturas agrícolas

Neste projeto serão usados modelos matemáticos que simulam diversos processos eco fisiológicos de culturas Agrícolas. Em geral, estes modelos trabalham dentro de um ciclo diários, em função de variáveis meteorológicas que englobam temperatura, umidade do ar, radiação solar, precipitação, nível de CO2 atmosférico, dentre outros, com base na localização geográfica a ser simulada. Alguns modelos também utilizam condições do solo, que podem envolver balanço hídrico e nível de nutrientes disponíveis e, por fim, o tipo de cultivar sendo simulada, no caso como variáveis que descrevem como ocorre seu desenvolvimento (STRECK et al., 2015). Os modelos que serão usados para estudo de caso em conjunto com a ferramenta desenvolvida no projeto são o Simanihot, SoySim e DSSAT.

O Simanihot é um modelo matemático dinâmico baseado em processos (*process-based model*). Foi projetado para trabalhar em ciclos diários e simula diversos processos eco fisiológicos da cultura da mandioca. O modelo foi desenvolvido pelo grupo Agrometeorológico da Universidade Federal de Santa Maria e é destinado a simular o crescimento, desenvolvimento e produtividade da cultura em questão no estado do Rio Grande do Sul, Brasil. O modelo utiliza como dados de entrada, as datas de plantio e de colheita, dados meteorológicos e balanço hídrico do solo, sendo um programa disponibilizado de forma gratuita (SIMANIHOT, 2016).

Outro modelo usado neste projeto, o SoySim, simula o desenvolvimento da cultura de soja e foi desenvolvido pela Universidade de Nebraska-Lincoln. Este modelo simula o potencial de rendimento, assim como as necessidades de irrigação, sem limitação pela irrigação e assumindo suplemento ideal de nutrientes e sem perdas de rendimento por influências do ecossistema, tais como granizo, ou de outros meios, tais como envenenamento por nitritos ou nitratos. O ciclo de simulação deste modelo também é diário (SOYSIM, 2016).

Já o DSSAT é um programa que engloba diversos modelos de simulação de culturas agrícolas, num total de 42 culturas. Oferece suporte ao manejamento de solo, clima e culturas assim como dados experimentais, por via também de utilitários e outros programas. Os modelos disponíveis simulam o crescimento, desenvolvimento e potencial como uma função de condições agrometeorológicas, tendo sido usado tanto no refino de manejo, em fazendas ou para análise de impacto climático sobre as culturas suportadas (DECISION SUPPORT SYSTEM FOR AGROTECHNOLOGY TRANSFER , DSSAT).

3 DESENVOLVIMENTO

Na sequência, serão descritas as tarefas desenvolvidas para se chegar no objetivo proposto por este trabalho. Na primeira etapa será realizada a concepção do software automatizador, onde serão analisados problemas de automação em softwares de simulação de culturas agrícolas e soluções existentes para automatização de interfaces gráficas, levantando os requisitos necessários para a solução dos problemas de automação destes softwares, dentro do objetivo proposto. Após a concepção Será modelada uma solução, procurando compreender que componentes são necessários e por qual motivo eles necessitam ser criados. Finalmente será realizado o desenvolvimento do software, verificando as diversas abordagens para criação dos componentes, discutindo as escolhas realizadas apos verificação das abordagens possíveis.

3.1 Concepção

Nesta etapa serão apresentados casos de uso obtidos com três atores, e posteriormente, uma análise com três ferramentas já existentes para automação de tarefas em programas com interface gráfica, o UiPath, o Sikuli e o Autoit, procurando embasar o objetivo deste projeto, os requisitos da solução e as decisões tomadas durante a modelagem e desenvolvimento do software.

3.1.1 Elaboração dos casos de uso

Com as reuniões realizadas semanalmente no primeiro mês, foram discutidos e coletados três casos de uso com estudantes da área de agrometeorologia no Departamento de Fitotecnia da UFSM, procurando entender as necessidades de automatização dos estudantes em cada um dos modelos com o qual trabalham.

O primeiro caso de uso é referente as necessidades da Engenheira Agrônoma Amanda thirza Lima, mestranda no programa de Pós-Graduação em Agronomia da UFSM, nas atividades de seu projeto de dissertação para indicar um novo zoneamento agroclimático para a cultura de mandioca no estado do Rio Grande do Sul utilizando o programa Simanihot, modelo de simulação da cultura da mandioca.

No segundo caso de uso foi tratado o objetivo da Jossana Ceolin Cera, Meteorologista e Doutora em Engenharia Agrícola pela Universidade Federal de Santa Maria, com objetivo

já realizado durante seu doutorado de forma completamente manual, onde foi usado o modelo SoySim para simular o crescimento, desenvolvimento e produtividade de Soja para o estado do Rio Grande do sul.

O último caso de uso foi referente a tese de Cesar Augusto Fensterseifer, graduado em Engenharia Ambiental, Mestre em engenharia civil e ambiental e atualmente aluno de doutorado do programa de pós-graduação em engenharia agrícola da UFSM, que deseja gerar previsões de safra de soja para auxiliar no planejamento agrícola do Estado do Rio grande do Sul.

Em todos os casos de uso as pré e pós condições são as mesmas, no caso, a ferramenta deve iniciar em condição de realizar uma automatização e finalizar em um estado capaz de iniciar novamente a mesma automatização. Os três casos de uso são descritos em ordem:

Caso de Uso 1: Automatização de simulações no modelo Simanihot para novo zoneamento agroclimático para a cultura de mandioca

breve descrição: Para o desenvolvimento da tarefa é preciso fazer muitas simulações para que se possa indicar os melhores dias de plantio em uma série histórica de anos agrícolas. Para que sejam realizadas essas indicações é necessário que sejam realizadas simulações diárias da data de plantio de primeiro de julho a 31 de dezembro para todos os anos desde 1960 a 2015, para 18 locais utilizando 4 cultivares; é necessário também fazer uma simulação para uma cultivar, em uma única data de plantio, local e ano especifico.

Fluxo Principal:

1. Os seguintes campos são preenchidos, mas não variam: concentração de dióxido de carbono (CO2) 400 ppm, densidade de plantio 15625 plantas por hectare, simulação a partir do plantio, data de colheita (15/06), modelo de balanço hídrico de *Thornthwaite* e *Mather* e respectivamente neste modelo de balanço hídrico, a profundidade de maniva 8 cm e profundidade máxima de raiz 30cm.

2. Será inserido um arquivo de entrada para cada local (Bagé, Bento Gonçalves, Bom Jesus, Caxias do Sul, Cruz Alta, Encruzilhado do Sul, Irai, Lagoa Vermelha, Passo Fundo, Pelotas, Porto Alegre, Rio Grande, Santa Maria, Santa Vitória do Palmar, Santana do Livramento, São Luiz Gonzaga, Torres, Uruguaiana). Para cada arquivo será utilizado um tipo específico de solo e será selecionado o local específico.

3. As simulações serão realizadas para 3 cultivares, sendo elas FEPAGRO RS13, Estrangeira e São José.

4. Para cada arquivo de entrada, serão realizadas simulações que variam de n = 1ř de agosto até 31 de dezembro, com variando de um em um dia.

5. A data de colheita não variará com as simulações, será sempre dia 15 de junho.

6. Depois de completar todos esses passo, finalmente clicará no botão SIMULAR.

Fluxo Secundário:

1. Nenhum

Fluxo de exceção:

1. Em caso de qualquer interação do usuário durante a tarefa de automação, pausar a simulação e esperar que o usuário recomece a automação ou cancele.

2. No caso de encontrar algum alerta de campo preenchido incorretamente, encerrar a automação e envia o usuário para a pós condição.

Caso de Uso 2: Automação de simulações no modelo SoySim para simular o crescimento, desenvolvimento e produtividade de Soja para o estado do Rio Grande do sul

breve descrição: Fazer simulações para 37 pontos espalhados no Rio Grande do Sul (37 arquivos de dados meteorológicos), com 120 anos de dados em cada arquivo, de 1980 a 2099, utilizando 7 datas de semeadura (01/08, 01/09, 01/10, 01/11, 01/12, 01/01 e 01/02) e 3 grupos de maturação (4.8, 5.5 e 6.0), em 2 cenários climáticos futuros. Dois conjuntos de dados com 37 arquivos de dados meteorológicos cada um. Dividindo cada um desses 37 arquivos em 3 arquivos com 45 anos de dados, pois o modelo SoySim não suporta fazer a simulação com um arquivo de dados tão extenso.

Fluxo Principal:

1. Inserir um arquivo de entrada para cada ponto (27$\underline{5}$1 19602007, 27$\underline{5}$1 20062053, 27$\underline{5}$1 20522099, 27$\underline{5}$2 19602007, 27$\underline{5}$2 20062053, 27$\underline{5}$2 20522099 e assim por diante).

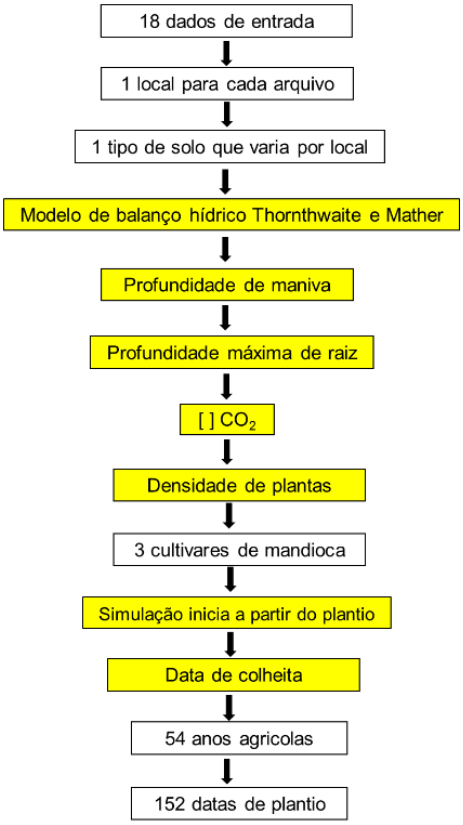


Figura 3.1: Fluxograma do caso de uso 1.

2. Realizar as simulações para os grupos de maturação (4.8, 5.5 e 6.0).

3. Para cada arquivo de entrada realizar simulações que variam de 01 de agosto até 01 de fevereiro, iniciando na semeadura, de um em um mês

4. Modificar a densidade de população de plantas para 315.

5. Depois de preencher estes campos, clicar no botão *run*.

6. Copiar os dados a partir da linha 14 a 59 do arquivo de resultado TmpOut.txt e armazenar todos em um só resultado filtrando itens específicos e gerar um arquivo de resultado com o mesmo.

Fluxo Secundário:

1. Reiniciar a automação alterando a automação para que a mesma não seja rodada na função de *multiple years* mas sim seja rodada na função de *single years*, onde cada ano precisa ser definido na interface.

2. Copiar a linha 35 do arquivo resultante TmpOut.txt e adicioná-la ao final de um arquivo de resultado único.

Fluxo de exceção:

1. No caso de encontrar algum alerta de campo preenchido incorretamente, encerrar a automação e envia o usuário para a pós condição.

2. Caso o modelo apresente um alerta rodando no fluxo principal o programa deve passar para o fluxo secundário

3. Caso o alerta seja recebido enquanto o modelo já está no fluxo secundário, passar para o ano seguinte da lista de anos da automação.

4. Em caso de qualquer interação do usuário durante a tarefa de automação, pausar a simulação e esperar que o usuário recomece a automação ou cancele.

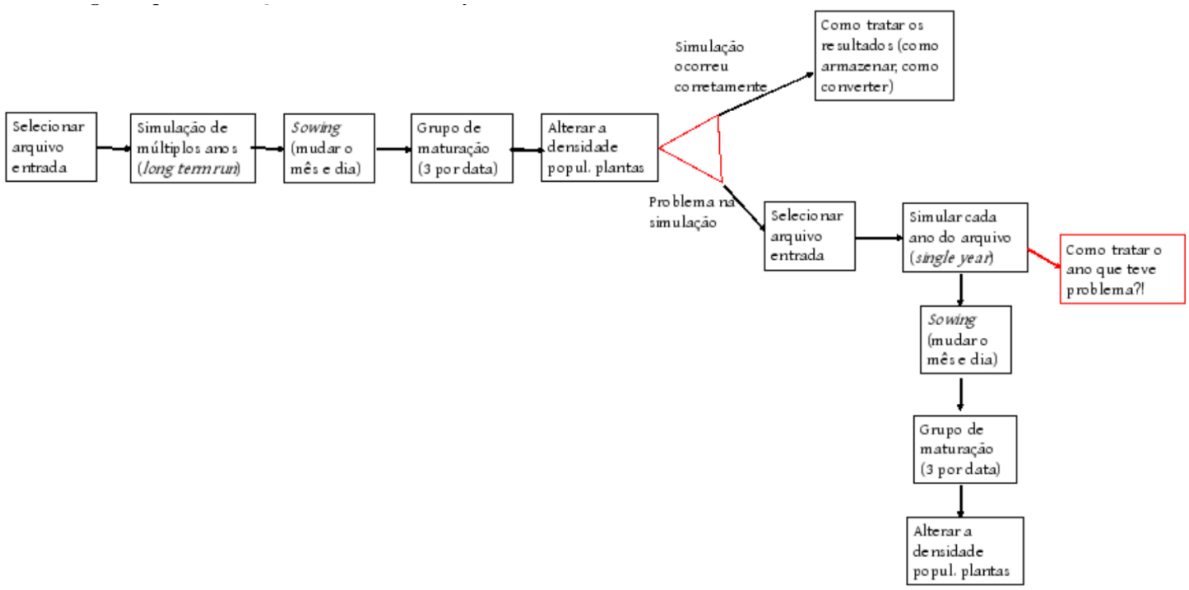


Figura 3.2: Fluxograma do caso de uso 2.

Caso de Uso 3: Automação de simulações no modelo DSSAT para gerar previsões de safra de soja para auxiliar no planejamento agrícola do Estado do Rio grande do Sul

breve descrição: O RS é um dos Estados brasileiros que mais sente os efeitos dos fenômenos El Nino e La Nina, e atualmente é o terceiro maior responsável pela produção de soja do País. É necessário gerar previsões de safra de soja para auxiliar no planejamento agrícola do Estado do Rio grande do Sul (RS). gerando várias simulações no modelo para diversas datas de semeadura.

Fluxo Principal:

1. Abrir o DSSAT, esperar carregar e clicar em *Crop Management Data*. A partir dessa ação, o programa XBuild inicializará, e o usuário deve então clicar em *New* (para começar a criar um experimento).

2. Criar um nome inexistente para o experimento. *Institute Code* (SM para UFSM), *Site Code* (RS se for dentro do RS), *Year* (2013), *Experiment Number* (99 datas de semeadura), por exemplo. Lembrando que o usuário é livre para escolher a forma de preencher as lacunas dessa etapa do experimento, de uma forma que o torne autoexplicativo. No campo *Crop* (selecione *Soybean*). As demais lacunas podem ser preenchidas com -99.

3. Nessa etapa o usuário clica em *Environment* e depois em *Fields*. Aqui basicamente deve ser informado qual a série meteorológica o experimento vai utilizar, se existe algum tipo de drenagem e as características do solo daquela região. Ao rodar o modelo para outro local, o usuário deve adicionar mais um *level* e repetir o preenchimento com a série meteorológica e as características do segundo local, e assim sucessivamente.

4. O usuário clica em *Management* e seleciona *Cultivars*, a tela sera redirecionada para uma seção onde deve ser selecionada a cultivar ou as cultivares que serão utilizadas nas rodadas do modelo. Lembrando que essas já devem estar devidamente calibradas pois caso contrário, o nome da cultivar não aparecera na lista de seleção.

5. Inserção da data ou datas de semeadura, data de emergência, o método utilizado, a forma de distribuição das sementes, a densidade de sementes na semeadura, a densidade de plantas na emergência, o espaçamento entre linhas adotado, a direção das linhas em relação ao Norte e a profundidade de semeadura utilizada para um ou vários experimentos. Lembrando que se o experimento possui alguma característica diferente, esse deve ser inserido como um Novo *Level*.

6. Determinar a data que o modelo vai começar a simulação do balanço hídrico ou de nutrientes no solo. É aconselhado começar o balanço hídrico no solo entre 20 e 30 dias antes da data de semeadura, buscando uma simulação mais precisa das condições do solo na hora da semeadura.

7. Selecionar os balanços que gostaria que o modelo realizasse em cada experimento. Aqui

pode ser ativado ou não o balanço hídrico, de nitrogênio entre outras. Esses detalhes podem aumentar o desempenho das simulações.

8. Decidir os métodos que serão utilizados nos balanços que foram selecionados no passo anterior. Se o balanço hídrico foi selecionado, aqui o usuário deve selecionar o método que o modelo deve utilizar para as rodadas em cada experimento. Essa regra também vale para os outros balanços que o usuário deseja que o modelo realize.

9. Caso o usuário tenha dados mais específicos ou tenha utilizado irrigação em algum dos experimentos que deseja simular, essa é a etapa em que vai inserir as características do manejo utilizado.

10. Seleciona as informações que serão exibidas no final das rodadas como: Crescimento de massa seca diária, balanço hídrico....entre outros.

11. Caracterizar cada experimento, identificando cada experimento que vai utilizar a primeira estação meteorológica por exemplo, ou o solo número 1 entre outras informações.

12. Após realizar essas etapas o usuário deve clicar no botão *Refresh*, que atualizará as novas informações e posteriormente pode abrir o programa DSSAT e começar a rodar os experimentos cadastrados.

Fluxo Secundário:

1. Nenhum

Fluxo de exceção:

1. No caso de encontrar algum alerta conhecido do modelo tais como alerta de temperatura máxima menor que a mínima, encerrar a automação e envia o usuário para a pós condição.

2. Em caso de qualquer interação do usuário durante a tarefa de automação, pausar a simulação e esperar que o usuário recomece a automação ou cancele.

3.1.2 Análise dos casos de uso em outras ferramentas de automação

Após a coleta dos casos de uso, foi analisado como outras ferramentas existentes atualmente, modelam os problemas observados nos casos de uso coletados, objetivando compreender como e sob quais condições, estas ferramentas conseguem solucionar os casos de uso.

Dentre as ferramentas escolhidas, foram selecionados três programas de automação de interfaces gráficas capazes de solucionar os casos de uso. Os três programas escolhidos foram descritos na revisão bibliográfica deste projeto, sendo estes, UiPath, Sikuli e Autoit.

Como forma de avaliação adicional do quanto estas ferramentas se mostram acessíveis aos domínios de conhecimento do usuário, foi proposto a um dos atores dos casos de uso, que este tentasse utilizar as mesmas ferramentas de automação descritas na revisão literária, para resolver seu caso de uso. Descrevendo a experiência de forma objetiva, em prós e contras, analisando se foi obtido sucesso ou não e onde os problemas responsáveis pelas falhas se originaram, caso encontrado algum.

UiPath:

Prós: O programa permite virtualmente a realização de qualquer tarefa por meio da construção de um fluxograma, com diversos elementos, vários deles autoexplicativos, tais como as regras de interação de clique e digitação na interface dos modelos matemáticos. Fornece lógica suficiente para as repetições e iterações necessárias nos casos de uso, também permite mover ou renomear arquivos para que as rodadas não se sobrescrevam no caso 2.

Contras: O programa é muito complexo, fator especialmente observado ao tentar modelar regras para o tratamento de erros no caso 1 e 2, contem muitas opções, elementos carregados e complexos e poucas funções realmente genéricas. Em outro exemplo de complexidade, alguns elementos envolvem expressões VisualBasic. Depende da chamada de métodos externos ao programa para tarefas não contempladas pelos elementos de fluxograma, fator necessário no *parsing* de valores em arquivos necessários para o caso 2, o que envolve métodos tradicionais de automação, com linguagens de programação convencionais, podendo recorrer inclusive ao *powershell*.

Sikuli:

Prós: O programa é fácil de entender com elementos autoexplicativos e não muito complexos, é capaz de realizar as tarefas básicas tais como interagir com a interface do modelo e esperar por eventos.

Contras: Para os casos de uso, os elementos visuais presentes não são suficientes e não contemplam repetições, trabalhar com arquivos, tratar erros, dentre outros, tornando necessário que o usuários recorra a linguagem Python para realizar a automação.

AutoIt

Prós: Novamente um programa capaz de realizar virtualmente qualquer tarefa. Apesar de sua abordagem bastante tradicional de programação em texto, possui uma biblioteca bem ampla de funcionalidades.

Contras: O programa é totalmente dependente de linguagem *script* tipo BASIC própria, sem nenhum elemento visual ou autoexplicativo. A automação se torna um desafio possivelmente tão grande quanto o problema original, já que o usuário precisa conhecer com certa profundidade a linguagem, o programa e as bibliotecas do AutoIt.

Análise do caso de uso 2 realizada pelo autor de forma direta nos 3 programas de automação sugeridos sem intervenção de terceiros ou qualquer pessoa do domínio da área:

Caso de uso 2 Analisado pelo ator

Automação de simulações no modelo SoySim para simular o crescimento, desenvolvimento e produtividade de Soja para o estado do Rio Grande do sul.

UiPath:

Prós: Conseguiu fazer com que o software rodasse o SoySim para *SingleYear* e *LongTermRun*. Uma rodada sozinha.

Contras: Não conseguiu inserir um *loop* para que múltiplas rodadas fossem feitas, também não conseguiu fazer com que o programa mudasse os arquivos de entrada automaticamente. Além disso, não conseguiu fazer com que o programa entendesse a janela de erro do SoySim.

Sikuli:

Prós: consegui montar o *scrit* que é no mesmo formato do UiPath (visual).

Contras: Na hora de rodar, o programa apresentou um erro que o ator não consegue interpretar, erro apresentado na figura 3.3. Desistiu da ferramenta devido ao erro.

AutoIt

O ator não conseguiu compreender minimamente como a ferramenta funciona, foi capaz de perceber que o programa usa código textual para automatizar as tarefas mas por não ter profundo contato com linguagens de programação ou BASIC, não foi capaz de encontrar uma solução intuitivamente ou em tempo hábil no manual do programa. Desistiu da ferramenta devido à dificuldade.

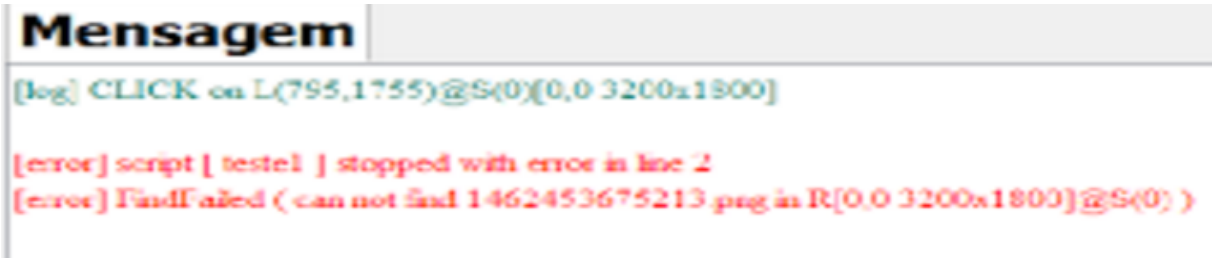


Figura 3.3: Erro obtido pelo autor no programa Sikuli.

Observando as informações obtidas ao analisar os casos de uso com outras ferramentas de automação, foram encontrados alguns pontos relevantes para a Proposta do projeto.

1. Linguagens de programação tradicional não contribuem para os atores solucionarem seus

problemas e inclusive dificultam a obtenção de uma solução em alguns casos.

2. Apesar de possibilitar a solução, teoricamente, de qualquer problema inclusive fora da automação de interfaces gráficas, linguagens de programação tradicional são desnecessariamente complexas para os atores e inclusive, para o problema deste projeto, aumentaram as dúvidas dos atores e problemas que eles precisavam considerar durante a obtenção de uma solução, tal como a validade sintática de suas soluções assim como coleta de informações em outras fontes com relação a vocábulos e semântica da linguagem.

3. Os momentos em que os atores mais se beneficiaram das ferramentas analisadas foi quando as mesmas ofereceram, componentes sucintos e autoexplicativos, ou seja, que descreviam o que fazem, como usá-los e como não usá-los.

4. O segundo momento em que os atores mais se beneficiaram das ferramentas analisadas foi quando os componentes oferecidos interagiam com os atores, impedindo maneiras incorretas de seu uso ou explicitando o uso correto por meio de diversas símbologias tais como o encaixe perfeito entre os elementos, dentre outros.

5. O elemento-chave para os atores residiu na capacidade ou não da ferramenta em explicar o que se pode fazer ou não, tomando para sí algumas tarefas que exigem atenção, como a análise em algum nível da validade da solução.

3.1.3 Coleta dos requisitos funcionais

Com base nas informações obtidas durante a análise prévia do problema, dos casos de uso, análise de ferramentas existentes, assim como discussão com os atores envolvidos, surgiram ideias de funcionalidades, das quais foram selecionados alguns requisitos funcionais mínimos para se chegar ao objetivo deste projeto assim como a solução dos problemas principais nos casos de uso.

1. Deve ser possível executar, salvar ou carregar automações construídas pelo ator de forma que se possa reutilizar estas automações, construídas anteriormente no programa. Também é preciso que seja possível descartar a automação.

2. Dois elementos, um capaz de executar a ação de clicar uma vez em uma área da tela e outro capaz de executar a ação de clicar duas vezes em uma área da tela; o ator deve

ser capaz de escolher a área da tela para cada instância do elemento e o elemento deve representar a área onde realizará a ação.

3. Um elemento capaz de esperar por uma alteração na tela ou evento que indique que a automatização pode prosseguir para as próximas tarefas definidas pelo ator.

4. Um elemento capaz de digitar textos nos campos do programa a ser automatizado, como se fosse o usuário digitando.

5. Elementos complementares que consigam expressar a lógica, condições, repetições ações e valores tais como números e textos.

6. Um elemento capaz de ler ou gravar um arquivo de texto de forma que se possa alterar o texto lido ou que se possa apenas copiar o arquivo para outro lugar, podendo mudar o nome.

7. Um elemento que permita tomar decisões que alterem o fluxo de automação.

8. Uma forma de pausar ou continuar a simulação.

3.2 Modelagem

Para que um software possa automatizar outro software com interface gráfica, é necessário que o software que realiza a automação possa enviar informação para o software a ser automatizado assim como sua interface gráfica. Analogamente, é preciso que o software automatizador possa receber informação do software a ser automatizado.

Possibilitar que o software automatizador envie e receba do software a ser automatizado, envolve estabelecer quais são as possíveis entradas e saídas do software a ser automatizado, sabendo que este desconhece a existência do automatizador, para em seguida definirmos as entradas e saídas do software automatizador, estabelecendo assim, nossa interface de comunicação.

Após conhecermos as entradas e saídas do software a ser automatizado serão modeladas as entradas e saídas do software automatizador, visando estabelecer uma interface com o software que queremos automatizar. Ao mesmo tempo, serão modelados os componentes internos que dão ao software automatizador, as condições necessárias para realizar as funcionalidades concebidas durante a coleta de requisitos.

A entrada e saída precisará também ser modelada considerando como o ator ou usuário irá interagir com o software automatizador para que este realize a automação necessária. A esquematização da interação entre o software automatizador, o usuário e o software a ser automatizado, pode ser observado na figura 3.4.

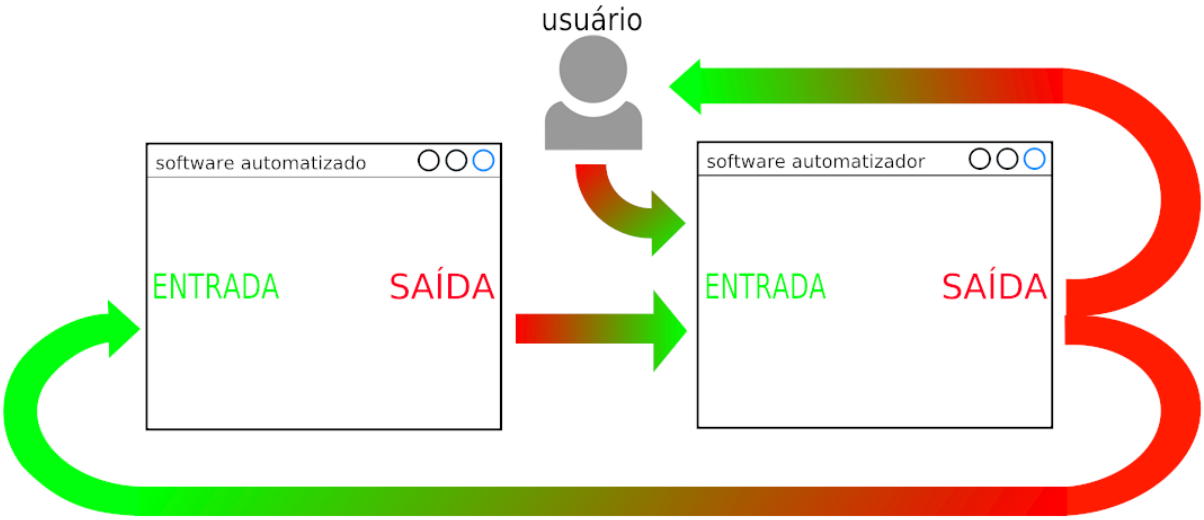


Figura 3.4: Descrição da interação entre o software automatizador e o software a ser automatizado, incluindo o usuário ou ator que utilizará o automatizador.

3.2.1 Entradas e saídas do software a ser automatizado

Para definir as entradas e saídas do software a ser automatizado[1], foram utilizados os modelos matemáticos de simulação de culturas agrícolas escolhidos para serem automatizados neste projeto.

Considerou-se quais eram os elementos com os quais se podia interagir, formas de interação e resultados observados. A figura 3.5 descreve as entradas e saídas observadas durante a análise dos softwares, sua natureza e componentes de destino ou origem. Essas entradas e saídas foram discretizadas em dois tipos de entrada e dois tipos de saída:

- entradas:

  ações: Interações que resultam em alguma saída do software, seja eventos ou dados.

  dados: Interações que fornecem dados ou valores, precedidos de uma ação, podendo resultar em um evento.

- saídas:

[1] Nesta seção, iremos nos referir a eles como software a ser automatizado ou apenas software.

eventos: Eventos ou respostas que buscam informar o usuário de alguma condição atingida no software, decorrente de alguma entrada, seja ação ou dado, podem esperar a interação do usuário.

dados: Dados e novas informações produzidas pelo software, produto final esperado pelo usuário, decorrente de algum evento.

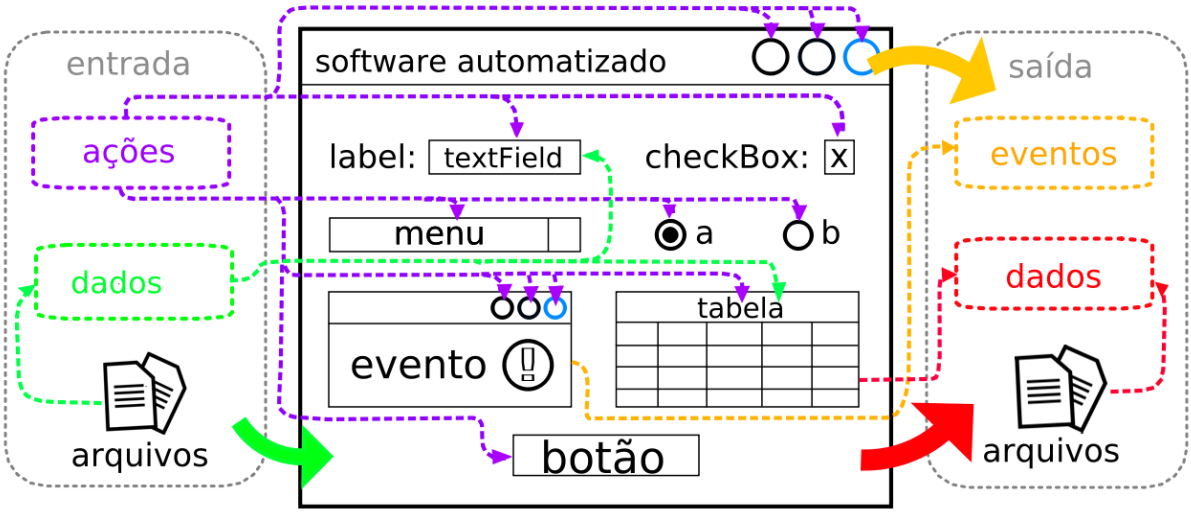


Figura 3.5: Descrição das entradas e saídas do software a ser automatizado, as grandes cetas representam o software como um todo, como destino ou origem, a verde mostra que é possível inserir arquivos no software, a vermelha, que é possível obter arquivos do software, e a laranja indica que, não somente mensagens, mas qualquer componente do software pode representar um evento, mudando de estado.

Como é possível observar na figura3.5, as ações correspondem a entradas que podem resultar em alguma saída, logo, podem produzir eventos ou dados como saída, podendo inclusive produzir ambos. As ações são consideradas entradas que ativam elementos ou funções do software.

Os dados quando considerados como entrada, podem ser inseridos de duas formas: após alguma ação, que indica que o dado pode ser inserido no componente foco da ação, como por exemplo, em um *textfield*, onde pode se inserir valores tal como números, texto ou a localização de arquivos, ou ainda podem ser inseridos automaticamente após alguma ação, caso o programa sempre procure por arquivos em algum local específico quando iniciado, onde a ação é a execução do software. Inserir dados resulta em evento, como por exemplo, mudança de estado do componente que recebe os dados ou uma mensagem.

Quando consideramos os eventos de saída, estes resultam, de alguma entrada. Os eventos procuram informar o usuário de alguma condição satisfeita por uma entrada, como por

exemplo, sucesso na inserção de um arquivo, ou de algum problema que deve ser observado, como por exemplo, algum campo preenchido incorretamente, ou que por exemplo, um elemento recebeu um evento e obteve foco, esperando pela inserção de dados.

Finalmente, considerando os dados de saída, podem ser observados duas formas: dados que são apresentados em componentes da interface gráfica, no caso dos softwares observados, podemos citar componentes como tabelas, gráficos e arquivos, que no caso dos softwares observados, são arquivos de texto. Os dados de saída são gerados por ações, quando gerados automaticamente, se deve ao fato de que o modelo tem a capacidade de acessar e carregar automaticamente resultados produzidos em algum momento no passado, estes dados são gerados após a inicialização do software ou seleção de simulação específica.

De forma resumida, do ponto de vista dos softwares observados, podemos observar que todas as saídas resultam de alguma entrada e inclusive a entrada de dados depende de alguma ação prévia, Incluindo aqui o ato de iniciar o software. Analogamente, podemos deduzir que toda entrada resultará em pelo menos uma saída, seja está um evento ou um dado.

Comportamentos diferentes foram considerados exceções, podendo resultar do fato de que o software parou de responder (receber entradas ou produzir saídas), ou a entrada fornecida resultou em saídas não esperadas pelo usuário.

3.2.2 Modelagem do software Automatizador

Feitas as observações de quais são as entradas e saídas do software que será automatizado e considerando que a automatização depende da interação entre ambos, é possível deduzir que a saída do automatizador[2], precisa conter elementos do mesmo tipo que a entrada do software a ser automatizado, direcionada a entrada do software a ser automatizado.

Analogamente, podemos deduzir que a entrada do automatizador precisa conter elementos do mesmo tipo que a saída do software a ser automatizado. Como o software automatizado desconhece a existência do automatizador, o automatizador precisa procurar e filtrar, quando necessário, saídas do tipo que o software a ser automatizado gera.

Além das entradas e saídas necessárias para estabelecer uma interface com o software automatizado, o automatizador também necessita estabelecer uma interface com o usuário.

Considerando o usuário e o software a ser automatizado, as entradas e saídas do automatizador foram discretizadas em três tipos de entrada e três tipos de saída:

[2] Nos referiremos ao software automatizador como automatizador ou software automatizador.

- entradas:

  ações: Interações vindas do usuário, que podem resultar em alguma saída do automatizador, seja eventos dados ou ações.

  eventos: Uma das saídas do software a ser automatizado, procurados quando necessário durante a automatização.

  dados: Quando inseridas pelo usuário, interações que fornecem dados ou valores para o automatizador realizar a automação, ou então, uma das saídas do software automatizado, procurados quando necessário durante a automatização. Podem resultar em algum evento no automatizador.

- saídas:

  ações: Ações geradas pelo automatizador durante a automação, destinadas a entrada de ações do software a ser automatizado.

  eventos: Eventos ou respostas que buscam informar o usuário de alguma condição atingida no automatizador, decorrente de alguma entrada do usuário, seja ação ou dados, podem necessitar a interação do usuário.

  dados: Podem tanto ser dados produzidos pelo automatizador durante a automação, destinados a entrada de dados do software a ser automatizado, como dados produzidos por alguma ação do usuário sobre o automatizador, que não são destinados ao software a ser automatizado.

Podemos observar na figura 3.6, que a entrada de ações por ser realizada pelo usuário, é destinada a componentes do automatizador considerados acessíveis ao usuário, podem ser necessários nos eventos gerados pelo automatizador para tomar decisões, nos elementos de automação para selecioná-los para montar a descrição da automação, assim como na descrição da automação para alterá-la, ocorrem também nas ações do software automatizador, quando o usuário, por exemplo, decide executar a automação. Geram alguma das saídas previstas no automatizador

A entrada de eventos ocorre seletivamente durante a automação, onde o software a ser automatizado é observado de acordo com regras definidas na descrição da automação em buscas de condições descritas pelo usuário na descrição da automação. Quando relevantes para a automação. Poderão resultar em alguma saída do automatizador, a depender da satisfação ou não da condição.

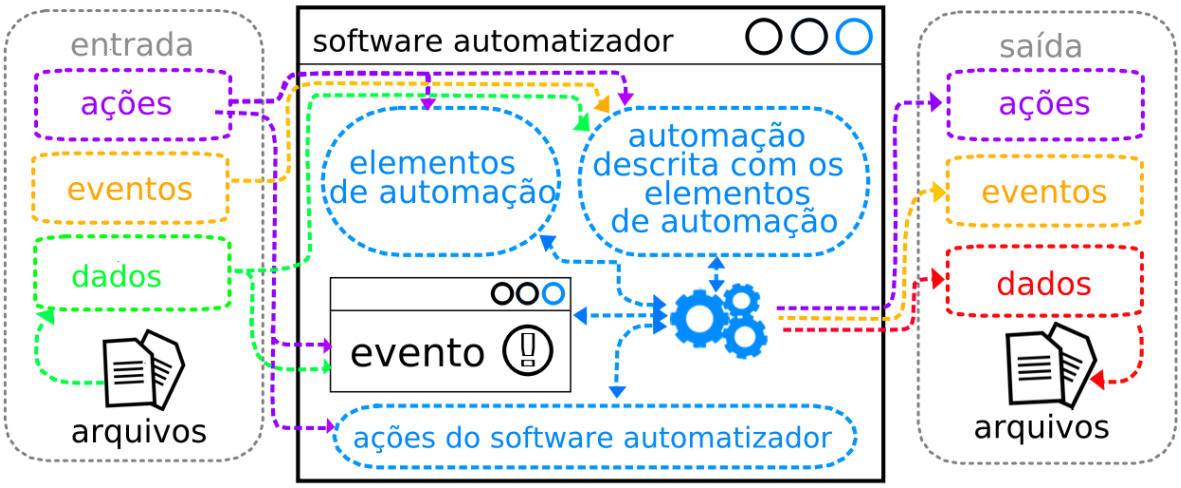


Figura 3.6: Descrição das entradas e saídas do automatizador, assim como da definição dos seus componentes internos. As engrenagens simbolizam os componentes de controle do automatizador.

Já a entrada de dados pode vir tanto do usuário como do software a ser automatizado. Quando do usuário, são destinados a descrição da automação ou eventos, caso contrário, são buscados seletivamente durante a automação, no software a ser automatizado, dependendo de como a automação foi descrita pelo usuário. Podem produzir eventos quando inseridos pelo usuário que podem necessitar atenção do mesmo. Também alteram ou geram saídas destinadas ao software a ser automatizado quando obtidos do software automatizado.

Na saída do automatizador as ações são produtos da execução da automação descrita com os elementos de automação, destinadas ao software a ser automatizado. Podem produzir eventos ou dados como saída no software automatizado que são do interesse do automatizador.

Os eventos, quando considerados como saída, são mensagens do automatizador destinadas ao usuário. São produtos de alguma entrada e indicam a satisfação ou não de alguma condição que necessita atenção, portanto, podendo necessitar a inserção de dados ou realização de alguma ação por parte do usuário.

Já a saída de dados pode ser tanto oriunda da execução da descrição da automação, destinadas ao software automatizador, como de ações realizadas pelo usuário, que não visam interagir com o software a ser automatizado, por exemplo, salvar a automação em um arquivo.

3.2.3 Componentes do software automatizador

Ainda podemos observar na figura 3.6, quatro componentes importantes para o automatizador, que permitem a realização das funcionalidades descritas nos requisitos.

Os elementos de automação são a coleção léxica da linguagem de automação, lexemas, unidades mínimas dotadas de significado lexical, componentes simbólicos com significado que descrevem alguma operação, condição, regra, elemento ou informação na automação. Estes elementos, por sua vez, são arranjados e conectados em ordem específica definida por suas propriedades, para formar a automação descrita com os elementos de automação.

Temos também um conjunto especial de ações do software automatizador que definem as ações do automatizador, o que deve ser feito com a automação descrita com os elementos de automação. Funcionalidades que dependem deste conjunto de componentes são, por exemplo, a capacidade de salvar automações ou ainda de executar a automação.

Todos os três elementos anteriores estão diretamente acessíveis ao usuário devido ao fato de que o usuário interage com eles, entretanto, apenas estes elementos sozinhos não são capazes de realizar e coordenar adequadamente suas funções e interações entre si, muito menos delegar os resultados destas funções e interações aos seus destinatários, seja este o software automatizado ou o usuário.

Quem faz este papel são os elementos de controle do automatizador que não estão diretamente acessíveis ao usuário, porém respondem as interações do mesmo com os outros elementos descritos, realizando as tarefas requisitadas pelos componentes assim como o envio e recebimento de informações entre eles que são pertinentes para as tarefas requisitadas, com o objetivo de concretizar as funcionalidades definidas na coleta de requisitos assim como o objetivo final, automatizar o software a ser automatizado.

São descritos na figura 3.6 como engrenagens e são os componentes responsáveis por controlar o automatizador, sendo responsáveis por diversas tarefas dentro do software automatizador, dentre elas:

- Carregamento de todos os elementos do automatizador com as propriedades e estados necessários na inicialização do automatizador.
- intercomunicação entre todos os componentes quando requisitado por algum componente por intermédio do usuário.
- realização das tarefas e dos processamentos requisitados pelos componentes a depender da funcionalidade que eles representam e dados que eles contêm.
- Apresentação de eventos e transmissão de informações dos mesmos para os outros componentes ou vice-versa.

- Coordenador na tarefa de utilização dos elementos de automação pelo usuário para serem usados na descrição da automação.
- realização das ações do software automatizador como executar, salvar, excluir ou carregar uma automação salva.
- Criação e delegação das informações de saída produzidas pelas tarefas descritas nessa lista.

3.3 Desenvolvimento do software

Com o software modelado é preciso criar os componentes idealizados, no desenvolvimento do software serão analisadas soluções para estes componentes, procurando as melhores soluções viáveis para satisfazer o objetivo deste projeto e requisitos elencados, embasando-se em observações feitas durante a concepção e modelagem para posteriormente, codificar o software, onde alguns trechos relevantes da codificação, serão apresentados e discutidos.

3.3.1 Interação com o software a ser automatizado

O primeiro problema observado durante a modelagem foi a necessidade de de interagir com o software automatizado, sendo que este não está ciente do automatizador. Foi identificado que o automatizador necessita enviar ações e dados para o software a ser automatizado, assim como precisa ser capaz de ler eventos e dados do software a ser automatizado.

Soluções observadas em outros softwares de automação testados durante o estudo dos casos de uso e revisão literária deste projeto envolviam duas formas de interação com o software a ser automatizado, utilização de APIs específicas do sistema, onde um exemplo de software analisado que utiliza este método como parte de suas funcionalidades, é o AutoIt, ou ainda, o método de captura e execução, método utilizado por outro software observado na análise dos casos de uso, o Sikuli ou Sikulix. A utilização de uma APIs ainda pode ser dividida em duas abordagens em sistemas Windows, o método baseado em *windows handles* (em sistemas UNIX, componentes similares seriam *file descriptors*) e o método baseado em *Windows Controls*.

Handles são identificadores únicos que fornecem encapsulamento abstrato e opaco para recursos internos do software que neste caso queremos automatizar, eles podem ser utilizados para identificar qualquer recurso dentro do software, indo além dos recursos observados apenas na interface, podem ser identificadores da janela do software, de um botão ou até mesmo,

componentes internos como listas, *threads* dentre outras estruturas e recursos utilizadas pelo software. Os *windows handles* são uma abstração particular dos recursos de um software em sistemas operacionais Windows. Em sistemas operacionais UNIX, uma abstração similar são *file descriptors*.

Windows controls por sua vez são também abstrações em sistemas Windows, são janelas filhos que o software usa em conjunto com outra janela que tornam possível a interação do usuário com o software. Em sistemas UNIX não existe uma abordagem padrão, já que alguns componentes de *frameworks* mais modernos podem ser acessíveis apenas na aplicação cliente e não no servidor X, isso quando consideramos que o servidor de vídeo em uso é o servidor X. Um exemplo de *framework* que permite acessar estes componentes em aplicações GTK é o ATK (ATK, 2014).

O método de captura e execução é baseado em captura e análise de imagens obtidas da tela do computador com o intuito de localizar elementos para posteriormente indicar uma interação padrão do usuário a ser realizada no elemento localizado, como ações de mouse e teclado. Outra abordagem ainda seria gravar a interação do usuário como movimentação do mouse e locais clicados e digitados. A abordagem é a mesma independente do sistema operacional.

Na tabela a seguir serão analisados os pontos positivos e negativos das três abordagens.

Windows handles
prós: • Permitem um nível bastante refinado de automação e em alguns casos permitem alterar profundamente o comportamento do programa ao mudar o comportamento de componentes e inclusive o estado das informações internas do mesmo. • É mais flexível e menos dependente de padrões que o método *windows controls*. Também oferece acesso a mais recurso e propriedades do software que o método *windows controls*. **contras:** • A utilização em geral só é prática em aplicações que o usuário conheça internamente o funcionamento, sendo relevante fazer um uso cauteloso já que *handles* são intencionalmente abstratos. *Handles* variam de seção para seção do programa e podem ser descartados durante a existência do programa ou simplesmente não representarem mais o mesmo elemento.

Windows controls

prós:

- Não depende da visibilidade dos *controls*. *Controls* que não estejam visíveis também estão ao alcance da ferramenta de automação já que a mesma tem acesso a todas as propriedades do programa.
- É possível alterar o comportamento do programa já que é possível definir qualquer estado possível para um *control* em específico.
- A interação do automatizador é mais próxima do contexto interno do programa, enviando, por exemplo, eventos diretamente para os *listeners* em botões, isso garante maior exatidão no comportamento o que é mais relevante nos casos em que se busca testar o softwares.
- Por não exigir muito processamento extra, análise complexa de dados e não depender da resposta da interface, é relativamente rápido.
- É uma abordagem mais estável que o método baseado em *windows handles* já que os *controls* são mais previsíveis e padronizados.
- Permite um grande reaproveitamento e reutilização em condições diversas se comparado ao método baseado em captura e execução.

contras:

- É limitado a *controls* conhecidos pelo software e API, exigindo um mapeamento de componentes customizados que diferem dos componentes tradicionais conhecidos pela API. Num exemplo, o software automatizador AutoIt apenas trabalha com os *controls* padronizados da Microsoft [3].
- Ainda exige um conhecimento relativo do comportamento do software a ser automatizado por parte do usuário, quais são os estados dos *controls* e qual é o significado dos estados do mesmo.
- Da mesma forma que no método baseado em *Windows handles*, alterar livremente as propriedades dos *controls* do software que se quer automatizar é perigoso já que pode levar a estados e condições inconsistentes dentro do software a ser automatizado, já que é possível definir qualquer estado possível para o *control* manipulado.

[3] esta informação pode ser encontrada na documentação do software AutoIt; https://www.autoitscript.com/autoit3/docs/intro/controls.htm

captura e execução
prós: • Não exige conhecimentos complexos de computação para utilização pelo usuário como compreensão de abstrações em nível de *frameworks* e APIs, como *handles* ou *controls* e como podem ser úteis em automação. • é de uso consideravelmente rápido para o usuário, pois não é preciso analisar tabelas ou listas de componentes em busca dos *handles* ou *controls* corretos. • pode ser usado em quaisquer condições, inclusive programas com interfaces não convencionais, contendo elementos fora do padrão, customizados. • independe do sistema operacional ou tecnologia da interface, pode controlar programas, páginas web, etc. **contras:** • Não é tão rápido como os outros métodos por exigir captura e análise complexa de imagens. • Não é possível elaborar um padrão formal menos abstrato para entrada de dados neste método devido ao fato de que as interfaces gráficas têm um dinamismo muito variado e complexo. • A dificuldade em estabelecer um padrão formal torna difícil reutilizar automações prontas já que o método é sensível a mudanças na interface ou no ambiente. • Mesmo em ambientes controlados, pequenas mudanças na interface do software a ser automatizado podem exigir alterações na descrição da automação.

Considerando as análises feitas das abordagens a cima, dos requisitos elencados e observações feitas na concepção do software a abordagem de captura e execução foi adotada embasado nos seguintes pontos:

- As outras duas abordagens, apesar de possuírem pontos positivos relevantes, beneficiam muito mais a automação de testes de softwares do que a automação que este projeto busca em softwares de simulação de culturas agrícolas, onde o acesso a propriedades e estados não previstos na interface gráfica do software não são pertinentes.

- A abordagem baseada em captura e execução é a que mais se aproxima do domínio de conhecimento dos usuários dos softwares que este projeto visa automatizar, já que o mesmo pode usar a abordagem com base nos conhecimentos que já obteve do uso da interface gráfica e do software. Esta proximidade pode ser observada inclusive nos requisitos cole-

tados, onde a discussão com os atores aborda o problema em nível de ações e interações que se encaixam na abordagem, como a utilização de teclado e mouse.

- As outras abordagens recorrem a conhecimentos peculiares da computação como orientação a objetos e modelos de abstração relacionados a este assunto.

- Apesar de a captura e execução ser computacionalmente mais lenta, a performance não é um componente crucial para a automatização dos softwares observados, já que apesar de ocorrer repetição, a repetição não é exaustiva como nos testes de software que busca investigar todo o comportamento do software, onde a performance do método pode ser relevante.

- Oferece maior flexibilidade para o usuário, já que pode ser usado em quaisquer condições, inclusive onde outras abordagens não são viáveis.

- É a abordagem que menos exige análise por parte do usuário, já que o mesmo não precisa procurar em tabelas e lista de componentes em busca de quais componentes estão relacionados com a tarefa que visa realizar no software, como encontrar qual elemento corresponde a um botão, ou campo de texto, ou funcionalidade do software a ser automatizado, ele pode utilizar os conhecimentos que já possui a respeito do software para realizar estas decisões.

- Oferece menor risco de expor para o usuário, a possibilidade de usar condições que não foram previstas pelo desenvolvedor do software que ele está automatizando.

Além das abordagens descritas, algumas interações do usuário mostraram permitir a utilização de métodos específicos, como por exemplo a execução do software. Onde o automatizador procuraria nas variáveis de ambiente pelo software a ser executado ou ainda realizaria uma busca em todo o sistema, ou ainda, o usuário poderia selecionas o executável.

Uma das vantagens deste método seria a execução de aplicações que não possuem um arquivo executável acessível ao usuário. Embora pertinente para a automação destes softwares, não foi observado vantagens para os softwares que este projeto objetiva automatizar, que por se focarem em usuário de programas com interface gráfica, disponibilizam executáveis que estão ao alcance do usuário por esta via.

Além das considerações feitas a cima, estas abordagens envolvem alguns conhecimentos formais de computação que devem ser observados pelo usuário. Nem sempre é garantido

que o software está devidamente registrado nas variáveis de ambiente como é possível observar no SoySim que não possui um instalador e não insere o programa na lista de caminhos de executáveis nas variáveis de ambiente do sistema operacional. Alguns sistemas operacionais, por exemplo, não expõem visivelmente as extensões de arquivos, o que poderia levar a ambiguidades quando o usuário informasse o nome do software a ser procurado, por considerando a extensão não relevante ou desconhecer sua existência, já que alguns softwares dentre os presentes no caso de teste divergem na sua natureza, podendo ser arquivos executáveis do Windows ou executáveis Java.

Outra consideração é que procurar o executável poderia se tornar uma tarefa excepcionalmente lenta caso fosse necessário vasculhar a árvore do sistema. E caso o usuário conhecesse a localização do executável, utilizar a abordagem de captura e execução seria igualmente eficiente para realizar a tarefa.

3.3.1.1 utilização da abordagem escolhida

Foi analisado como a abordagem de captura e execução poderia ser utilizada no software automatizador e optou-se por uma abordagem onde o usuário seleciona o elemento de automação e posteriormente define no elemento, a área sob a qual ele deve agir, com base em imagens como definição de local, é possível observar uma descrição abstrata da solução na imagem 3.7.

Nesta abordagem, as imagens capturadas do software a ser automatizado, seriam os eventos gerados pelo software a ser automatizado, quando torna a interface visível para interação, imagens estas capturadas pelo usuário. Após a localização do elemento, o automatizador pode enviar ações (controlando o dispositivo de entrada apontador, mouse), que gerarão um evento de obtenção de foco ou ativação no elemento, objeto da ação, também poderá enviar dados para os elementos em foco (controlando o dispositivo de entrada de texto, o teclado);

Foi observado também como um dos software analisados nos casos de uso o Sikulix ou Sikuli, utiliza esta abordagem. No caso do software observado, quando o usuário escolhe um elemento de automação que deve levar em consideração o componente do software a ser automatizado, ao qual é direcionado, o software automatizador se torna invisível, a tela escurece e uma mensagem pede para que o usuário selecione com o mouse uma área retangular onde o elemento de automação deve realizar a ação, posteriormente confirmando apertando *enter*.

Por fim, decidiu-se por uma abordagem similar onde o usuário primeiro seleciona o elemento de automatização, posteriormente interage com o mesmo para capturar a área onde

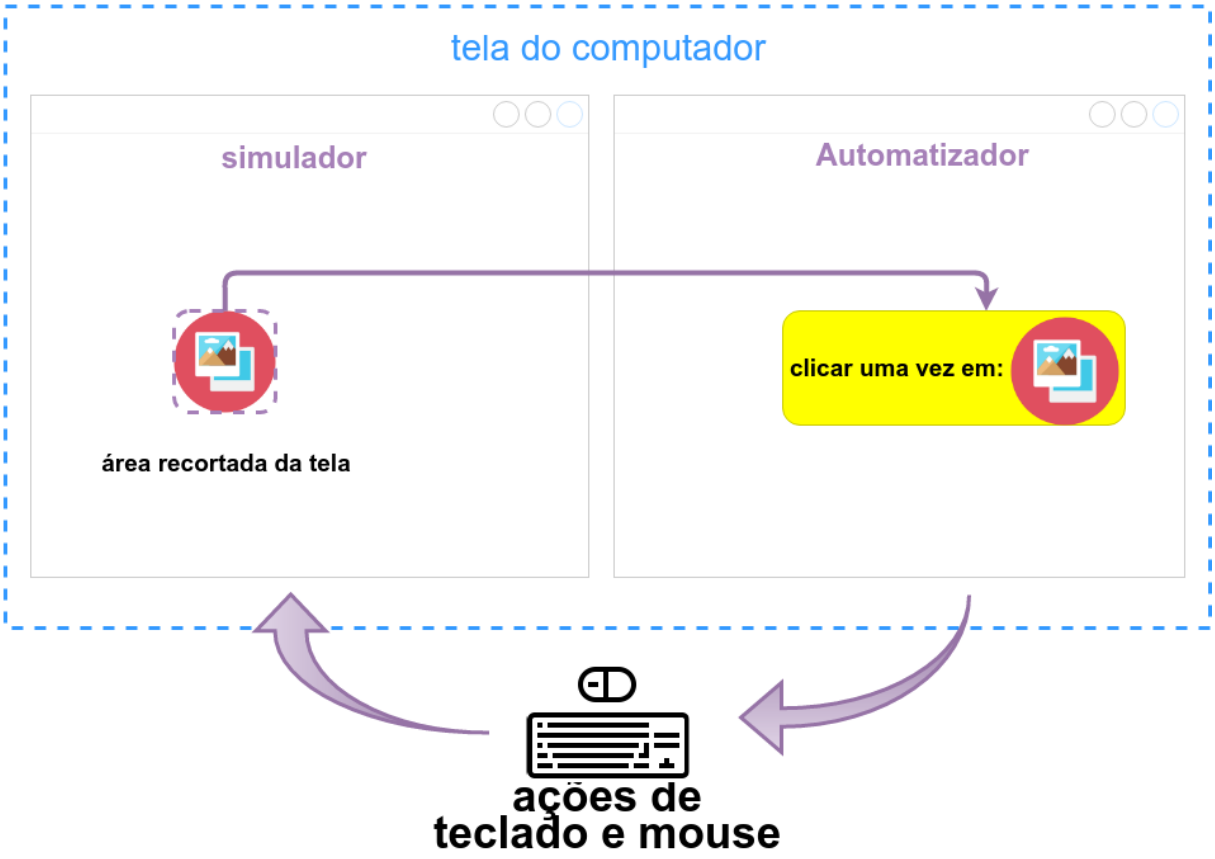


Figura 3.7: Estrutura geral da solução.

a automação deve ser realizada, com a diferença de que primeiro o usuário é alertado de que pode procurar, movendo janelas ou alterando a interface, pelo componente que quer capturar. Quando encontrado, o usuário realizará um comando de teclado onde a tela será capturada e o usuário poderá selecionar na captura a área foco da interação, confirmando igualmente com um *enter*.

3.3.2 Descrição da automação e seus elementos

Na modelagem foi definido dois componentes que tratam da construção da descrição da automação, a descrição da automação em si e os elementos de automação. O objetivo geral deste projeto trata em parte da realização de automação de tarefas em interfaces gráficas por meio de uma abordagem visual, por esta se aproximar do domínio de conhecimento do usuário, utilizando a biblioteca de programação visual Blockly.

Durante a concepção do software, foi observado e e justificado que, a programação tradicional dificulta e não complementa a automação para estes usuários por não permitir uma abordagem quase que totalmente baseada em lógica e nos conhecimentos que o usuário já detêm do

software a ser automatizado, embasando a troca de abordagem de linguagens tradicionais para linguagens visuais, demonstrando que estas diminuem a quantia de detalhes e conhecimentos com quais o usuário deve lidar.

Outra parte do objetivo geral consiste em utilizar a biblioteca de programação visual Blockly. A escolha da biblioteca se deu por alguns motivos que a tornaram adequada para o desenvolvimento deste software:

- Oferecia objetivamente suporta a diversos idiomas, por ser uma propriedade fundamental dentro do contexto de conhecimentos dominados pelos usuários, ou seja, o idioma em sí do usuário.

- Permite abordar a programação em um paradigma imperativo, procedural podendo ser estruturada, paradigmas mais próximo de como o usuário raciocina a solução de tarefas.

- É uma biblioteca voltada para desenvolvedores, de código aberto, permitindo a exploração, adaptação e expansão do código original oferecendo a flexibilidade necessária para a solução do objetivo e doa requisitos elencados de uma forma viável e gratuita.

- Não é uma linguagem de programação em si, muito menos de aplicação especializada, oferecendo componentes genéricos que permitem a construção de programas e resolução de problemas diversos em uma abordagem de programação visual desacoplada de uma linguagem tradicional, que posteriormente pode ser traduzida para uma linguagem tradicional, o que a torna bastante flexível, permitindo a abordagem de problemas como a automação de uma forma mais abstrata e especializada.

- Poucas limitações arquiteturais, podendo ser utilizada em conjunto com outras linguagens facilmente por ser uma biblioteca injetável em componentes web e permitir a tradução dos quebra-cabeças para diversas linguagens, oferecendo uma gama mais flexível de escolhas para a solução de outros componentes e problemas observados no desenvolvimento.

A biblioteca sera utilizada para definir um vocabulário da elementos de linguagem visual, os elementos de automação, de forma a oferecer componentes que permitam expressar a lógicos necessária para definir as repetições, ordem das operações assim como a criação de elementos que expressem as ações de automação elencadas nos requisitos, dentro da abordagem de captura e execução. A biblioteca irá se encarregar de fazer algumas validações da montagem,

verificando a validade ou não dos encaixes entre os elementos e impedindo que elementos não compatíveis sejam conectados.

também será utilizada para traduzir a descrição para uma linguagem de programação tradicional que será usada pelo automatizador para rodar as automações, assim como oferecer uma área que permita montar a automação. Também através da biblioteca será oferecida algumas funcionalidades do software abordadas nas ações do software automatizado. A figura 3.8 descreve de forma resumida, levando em consideração apenas as saídas relevantes e a integração do Blockly exemplificando com algumas ações do software automatizador.

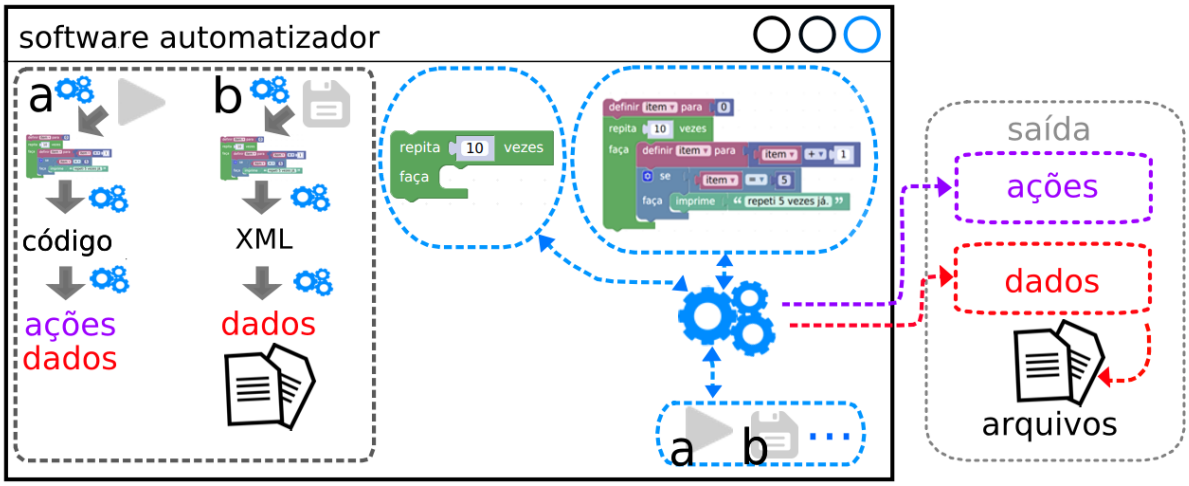


Figura 3.8: Descrição da interação do Blockly com algumas ações do software automatizador.

Além dos diversos blocos já existentes na biblioteca, úteis para a definição de repetições, variáveis como texto números e listas, lógica e operações, para adaptar o Blockly ao contexto de automação, serão desenvolvidos mais 4 blocos fundamentais para a automação, observando os requisitos elencados, dentro da abordagem de captura e execução já discutidos:

- Um bloco que permite realizar clique simples em uma área da tela.
- Outro que permite realizar clique duplo em uma área da tela
- Um que permite esperar por um evento dentro do contexto de captura e execução, procurando a cada tanto intervalo de tempo definido pelo usuário
- Por último, outro bloco que permita digitar valores que possam ser convertidos para texto, como números, texto em si ou variáveis cujo valor possam ser convertidas para texto.

As ações do software automatizador, observadas em parte na figura 3.8 são um dos elementos concebidos durante a modelagem, necessários para que o usuário possa dizer ao automatizador, o que deve ser feito com a descrição da automação com o objetivo de fornecer as funcionalidades previstas no ponto 1, nestes componentes estão previstas as funções de salvamento, carregamento, execução e exclusão da automação.

Para todas as funções serão utilizadas capacidades existentes na biblioteca Blockly, que é capaz de salvar a descrição de um quebra-cabeças já montado ou carregá-lo e permite também a exclusão de quebra cabeças já montados. Para executar a automação a descrição será traduzido para uma linguagem de programação específica por meio da biblioteca Blockly e posteriormente o código gerado da automação será interpretada e executado.

3.3.3 Codificação

Como escopo foram delimitados alguns pré-requisitos necessários para codificar o software modelado dentro das abordagens já tratadas, onde, o software deve ser uma aplicação desenvolvida usando Java e JavaScript. Devido ao fato de que a biblioteca Blockly é desenvolvida em JavaScript e também, levando em consideração que a linguagem Java é uma linguagem robusta para lidar com a abordagem de cliente/servidor em aplicações web permitindo melhor integração com a biblioteca Blockly que por sua natureza executa em ambiente web como uma aplicação cliente, portanto, o automatizador trabalhará como um servidor que rodara, controlar e apresentar uma página web com a *workspace* da biblioteca Blockly injetada como uma aplicação web.

3.3.3.1 Integração do Blockly em uma aplicação Java

Em um primeiro momento, com o objetivo de rodar o Blockly, uma biblioteca JavaScript 100% *client side* dependente de um ambiente web, em uma aplicação java, optou-se por uma solução que envolve uma aplicação JavaFX por fornecer uma *web engine* com suporte para JavaScript, mais um cliente *web* embutido, descrita de forma geral na imagem 3.9.

O software ao iniciar carrega a página da aplicação web desenvolvida com base na biblioteca Blockly em uma instância da classe `WebEngine`, contida na plataforma JavaFX, sendo que esta instância pertencente a um objeto da classe `WebView` também oriunda da plataforma JavaFX, responsável pela gestão e renderização da página *web* na interface gráfica de uma aplicação JavaFX.

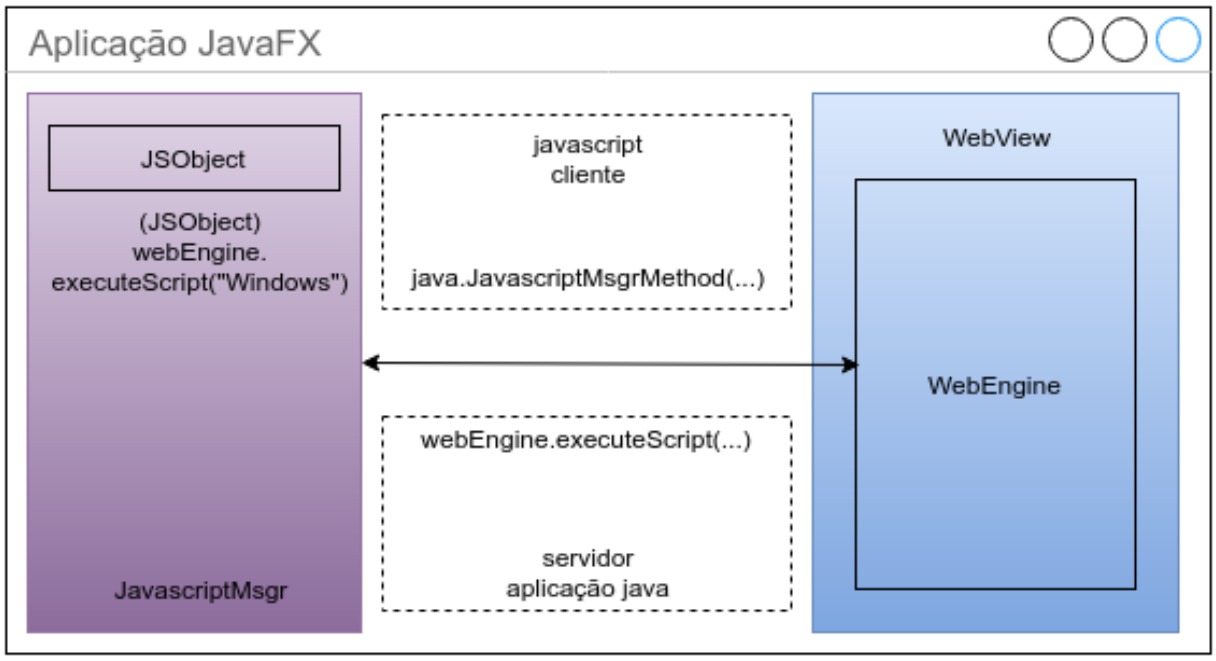


Figura 3.9: Estrutura geral da solução cliente/servidor.

Foi utilizada a funcionalidade LiveConnect, presente na maioria dos navegadores web, que permite a comunicação entre o navegador e aplicativos java para conectar o software e a aplicação web. Foi desenvolvida a classe `JavascriptMsgr` com o objetivo de estabelecer os métodos de comunicação entre a aplicação web e o software, contendo uma referência para o objeto instanciado da classe `javafx.scene.web.WebEngine` com o qual cria um objeto `netscape.javascript.JSObject` que torna a classe `JavascriptMsgr` visível para a aplicação *web*.

A classe `JavascriptMsgr` trata tanto as mensagens vindas da aplicação *web* como as mensagens enviadas do software, a imagem 3.10 apresenta dois exemplos de métodos presentes na classe `JavascriptMsgr` em trechos distintos[4], onde o primeiro envia uma mensagem para a aplicação web e o segundo recebe uma mensagem.

A classe possui diversos métodos que estendem as funcionalidades da aplicação *web* e permitem ao software utilizar a biblioteca Blockly para fornecer a biblioteca visual, gerar e executar código.

3.3.3.2 Interação e pesquisa por regiões na tela do sistema operacional

A interação com o Sistema operacional é realizada por instâncias da classe `java.awt.Robot` capaz de controlar os a entrada de teclado e *mouse*, gerando eventos nativos do sistema operaci-

[4] código completo disponível em https://github.com/RomuloPBenedetti/TCC/tree/master/Software/src/

```
1  // método que envia uma String contendo o arquivo XML gerado na última vez em que foi
2  // salvo a Blockly.mainWorkspace
3
4  public void loadBlocks(String xml) {
5      com.sun.javafx.application.PlatformImpl.runAndWait(() ->{
6          String escaped = StringEscapeUtils.escapeEcmaScript(xml);
7          engine.executeScript("loadBlocks(\""+ escaped +"\")");
8      });
9  }
10
11 // método que recebe um pedido para esperar enquanto não localizar uma imagem na
12 // pasta raiz, recebe um endereço relativo na pasta raiz do software e um valor
13 // de intervalo em milissegundos que deve esperar a cada procura sem sucesso
14 public void waitImg (String imgSrc, int milisec){
15     try {
16         BufferedImage target = null;
17         target = ImageIO.read(new File(imgSrc.substring(3)));
18         BufferedImage screenshot =
19             robot.createScreenCapture(primaryScreenBounds);
20         ImageSearch is = new ImageSearch(target);
21
22         while(is.search(screenshot)[0] == -1){
23             screenshot = robot.createScreenCapture(primaryScreenBounds);
24             System.out.println("tried");
25             Thread.sleep(milisec);
26         }
27         System.out.println("match!");
28     } catch (IOException | InterruptedException e) {
29         e.printStackTrace();
30     }
31 }
```

Figura 3.10: Dois métodos da classe mediadora da comunicação entre a aplicação web e o software

onal. Apesar de ter sido desenvolvida com o objetivo principal de realizar testes automatizados e aplicações autoexecutáveis, por ser capaz de manipular *mouse* e teclado em nível de sistema, pode ser utilizada para qualquer tarefa que busque automatizar o controle do sistema operacional.

Para encontrar as áreas a serem clicadas ou regiões a serem observadas na espera de uma mudança, foi desenvolvida a classe `ImageSearch` com uma solução própria de pesquisa otimizada para imagens e desempenho, que analisa uma imagem capturada da tela do sistema em busca da primeira referência exata da área escolhida pelo usuário.

Diversas outras soluções foram analisadas como a biblioteca OpenCV (OPENCV, 2016) (que inclusive é utilizada pelo automatizador Sikuli) ou ainda o *framework* brasileiro de processamento de imagens em Java, Marvin (ARCHANJO; ANDRIJAUSKAS; MUÑOZ, 2008). Porém outras soluções não apresentavam vantagens que melhorassem a abordagem de procura pelas regiões e traziam diversos problemas que não se justificavam, variando da necessidade de mais bibliotecas, adaptação a uma abordagem mais complicada e distante da necessidade do projeto ou ainda limitações em termos de plataformas.

A abordagem do Sikulix, por exemplo, permite a procura por imagens não somente exatas, mas similares, em níveis percentuais de similaridade configuráveis, utilizando o método `matchTemplate()` presente na biblioteca OpenCV. A abordagem também é mais rápida pelo fato de a biblioteca OpenCV ser uma biblioteca nativa.

Apesar de o método aumentar a flexibilidade de localização das áreas permitindo variações discretas, não apresentou resultados seguros no reconhecimento variações comuns nas automações, como mudança de campos de texto que fossem capturados sem texto e em algum momento durante a automação, fossem preenchidos, como é possível observar na figura 3.11.

Outro problema observado é que a utilização da biblioteca necessita instalação já que a biblioteca utiliza binários nativos para o sistema operacional do usuário. O que no caso do java, impede uma abordagem simples de distribuição de uma única biblioteca que possa ser carregada dinamicamente dificultando a portabilidade.

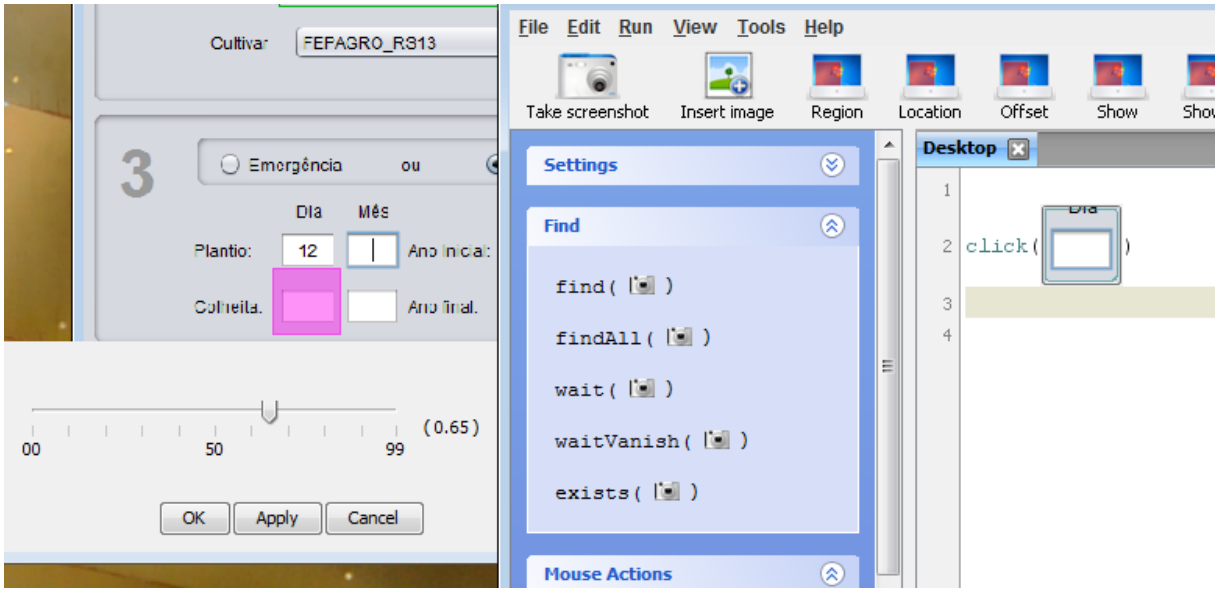


Figura 3.11: Imprecisão observado no método de procura em imagens do software Sikuli. `ImageSearch`.

A abordagem desenvolvida para o automatizador é descrita ilustrativamente na figura 3.12, é extraído da área selecionada pelo usuário, dois *pixels*, um com o maior valor de cor RGB dentre todos e outro com o menor valor dentre todos, assim como a distância 'x' e 'y' destes dois *pixels* da origem da região. Por questões de otimização os componentes RGB dos *pixels* da imagem são analisadas em conjunto, em um único inteiro de 32 *bits* tratado como um inteiro sem sinal para obter um valor absoluto, garantindo que o valor vá do maior ao menor valor possível para uma imagem com 8 *bits* por canal.

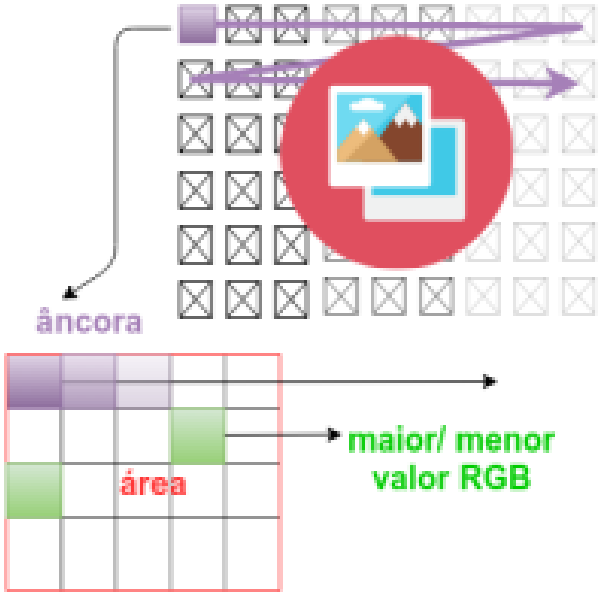


Figura 3.12: descrição visual da tarefa realizada pela classe. `ImageSearch`.

Posteriormente percorre todos os *pixels* pertinentes da imagem capturada da tela do sistema, de x = 0 até x = ((largura Da Captura) - (largura Da região)), usando a posição em que se encontra como âncora ou origem da região, para verificar nas posições salvas dos *pixels* extraídos da região se são iguais aos *pixels* na posição correspondente da imagem em relação a âncora, por meio das distâncias salvas com relação a origem da região.

Caso os 2 *pixels* sejam iguais, inicia-se uma análise de uma malha esparsa de *pixels* na região candidata, onde é somado a diferença dos *pixels* da região e a imagem capturada da tela, caso a diferença no final seja igual a zero, significa que foi encontrado uma área na imagem capturada igual à imagem da área que o usuário escolheu e neste caso a função retorna a posição 'x' 'y' da âncora.

Trechos relevantes da classe `ImageSearch`[5], podem ser observados nas figuras 3.13 e 3.14.

O método `StoreTarget` na figura 3.13, percorre a imagem que o usuário selecionou para ser clicada ou analisada e compara o valor de cada *pixel* com variáveis auxiliares, onde posteriormente sobrescreve o valor destas variáveis se o *pixel* for realmente um valor maior ou menor do que os considerados anteriormente.

Já na figura 3.14, pode ser observado um trecho do método `search` que procura pelo máximo e mínimo na imagem capturada da tela do sistema durante a execução da automação. Quando encontra a área que contem nas mesmas posições o valor máximo e mínimo obtidos

[5] código completo disponível em https://github.com/RomuloPBenedetti/TCC/tree/master/Software/src/

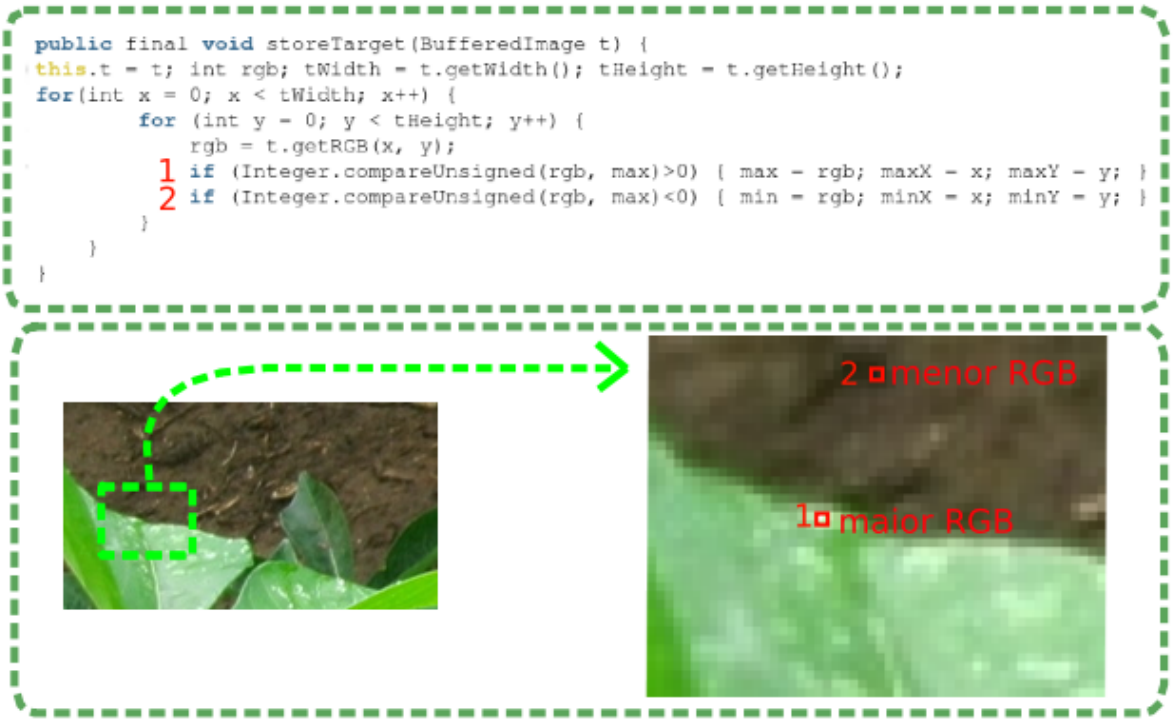


Figura 3.13: descrição do algoritmo de procura pelos máximos e mínimos na região, contido na classe `ImageSearch`.

pelo método `StoreTarget`, inicia uma analisa mais complexa da área candidata, um pixel a cada três. Caso a soma de todas as diferenças da área candidata resulte em um valor igual a zero, todas as *threads* fazendo a busca são paradas e a posição do *pixel* âncora ou origem da área candidata é retornado.

A tarefa foi codificada para dividir a busca entre os núcleos do processador do usuário para obter melhor performance, o método apenas retorna quando o todas as *threads* iniciadas pela pesquisa terminarem.

3.3.3.3 Utilização e adaptação do Blockly para automação

Durante a codificação foi necessário desenvolver os blocos de automação e seus geradores de código. É possível observar a descrição em JavaScript de dois blocos na figura 3.15 e a descrição formal em JavaScript destes blocos na figura 3.16[6].

Como é possível ver o bloco `click_doble_special` utiliza um campo próprio chamado `FieldImageButton`[6], desenvolvido com a finalidade de oferecer um campo semelhante ao campo `field_image`[7] presente na biblioteca Blockly, expandindo as funcionalida-

[6] código completo disponível em https://github.com/RomuloPBenedetti/TCC/tree/master/Software/src/
[7] o código desta classe pode ser encontrado em https://github.com/google/Blockly/

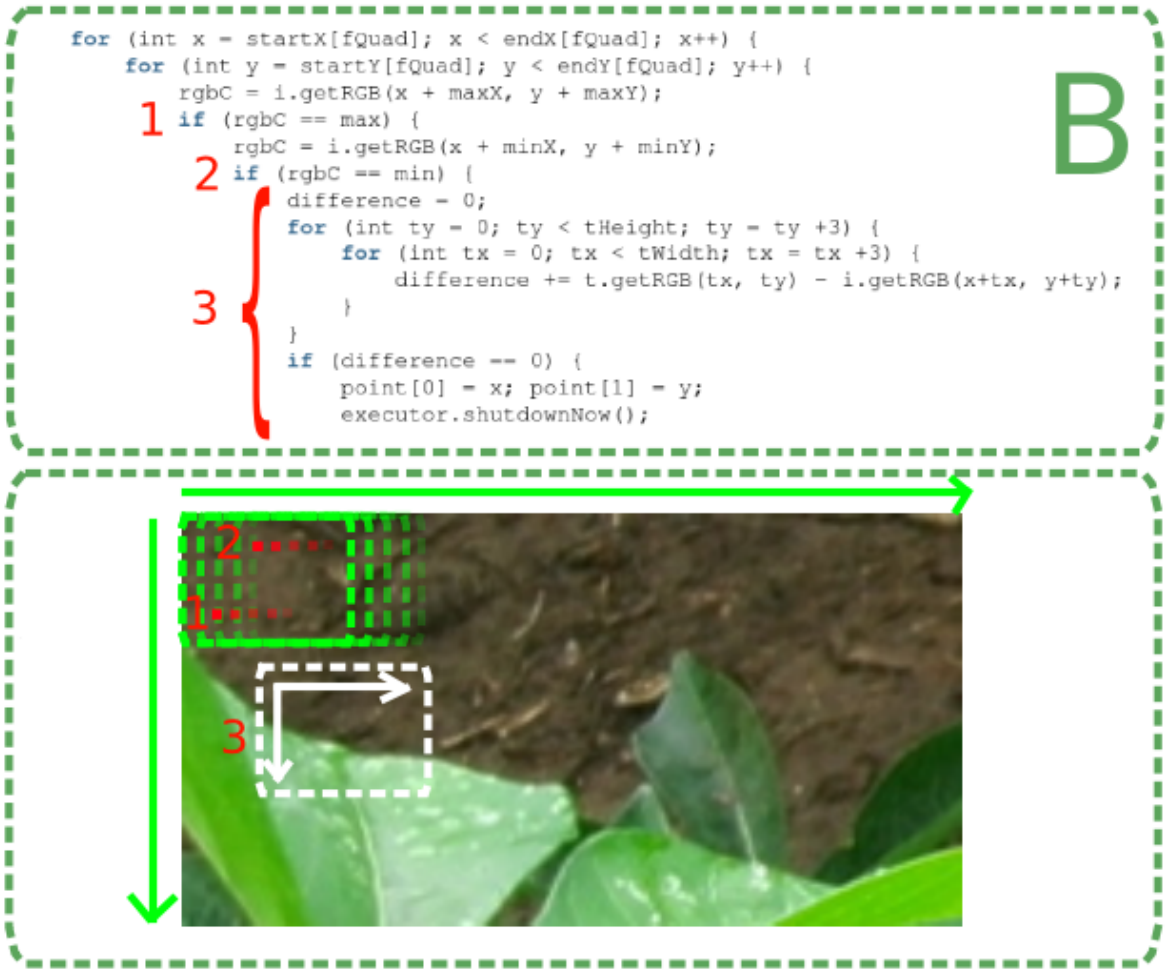


Figura 3.14: descrição do algoritmo queprocura pela região na imagem capturada, utilizando máximos e mínimos, contido na classe `ImageSearch`.

des disponíveis para a descrição de um bloco. Este campo adiciona à imagem um *eventListenner* que ao ser clicado envia uma mensagem que dá início a tarefa de captura de tela no software, que ao ser terminada pelo usuário, envia uma mensagem contendo o caminho da imagem salva da região selecionada pelo usuário que deve sofrer a ação do bloco junto com suas dimensões. Por sua vez a aplicação web atualiza a imagem do bloco para representar a área que o usuário escolheu.

Toda a tarefa é mantida coesa transferindo junto com estas mensagens o ID único que a própria biblioteca Blockly gera para cada bloco, assim tanto a aplicação web como o software sempre sabem para que bloco é destinada cada mensagem que também é salva com um nome único para seção.

Para o código gerado para o bloco, por questões de flexibilidade e simplicidade foi utilizada a linguagem JavaScript, já que é uma das linguagens nativas em uso no automatizador

```
Blockly.Blocks['click_single_special'] = {
  init: function() {
    this.appendDummyInput("imageInput")
        .appendField("clicar uma vez em:")
        .appendField(new Blockly.FieldImageButton("../images/icons/clickBlack.png", 30, 30,
            "", imageButtonEvent), "FieldImageButton");
    this.setPreviousStatement(true, null);
    this.setNextStatement(true, null);
    this.setColour(60);
    this.setTooltip('Procura na sua tela pela região que a imagem representa e clica no
        centro uma veze, para escolher, clique na imagem branca do bloco, vá ao local
        desejado aperte ctrl+shift+c, selecione onde clicar com o mouse e aperte enter.');
  }
};

Blockly.Blocks['text_typer'] = {
  init: function() {
    this.appendDummyInput("textToType")
        .appendField("digitar:")
        .appendField(new Blockly.FieldTextInput("texto"), "text");
    this.setPreviousStatement(true, null);
    this.setNextStatement(true, null);
    this.setColour(60);
    this.setTooltip('');
    this.setHelpUrl('http://www.example.com/');
  }
};
```

Figura 3.15: Descrição do bloco de clique e do bloco de digitação na aplicação web e seus respectivos geradores de código

```
Blockly.JavaScript['click_single_special'] = function(block) {
  src = (block.getFieldValue('FieldImageButton')).replaceAll("\\\\","\\\\");
  running = 'if (running) {\n';
  calling = "java.clickIn(\"" + src + '\",' + 1 + ');\n';
  end =     '}\n';
  var code = running + calling + end;
  return code;
};

Blockly.JavaScript['text_typer'] = function(block) {
  text = block.getFieldValue('text');
  running = 'if (running) {\n';
  calling = "    running = java.type(\"" + text + '\");\n';
  end =     '}\n';
  var code = running + calling + end;
  return code;
};
```

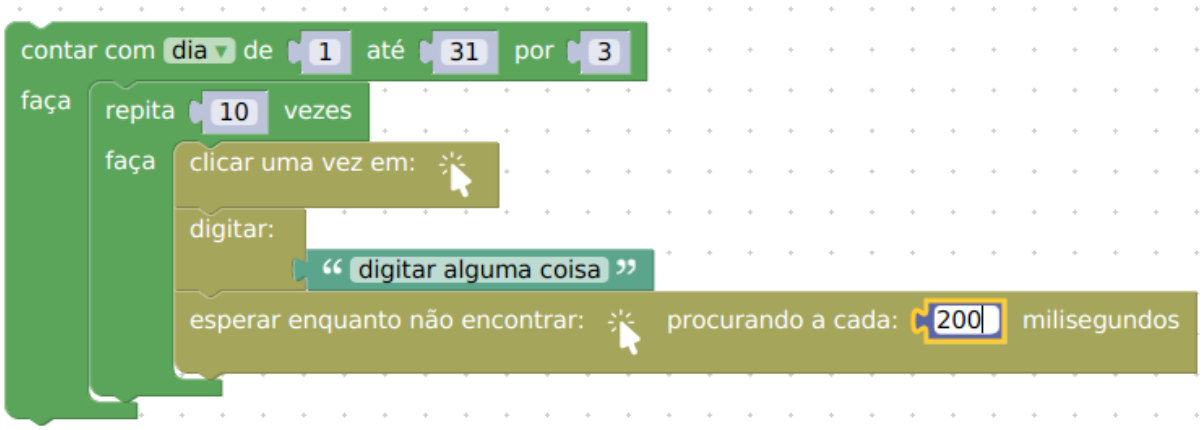
Figura 3.16: Descrição do bloco de clique e do bloco de digitação na aplicação web e seus respectivos geradores de código

e está disponível para os outros blocos da biblioteca por padrão.

Quando o usuário clica no botão para executar o modelo no software, é enviada uma mensagem para a aplicação web, pedindo que a biblioteca Blockly monte o código com base no quebra-cabeça, descrição da automação, e por fim, que a `WebEngine` execute o código com a função `eval` disponibilizada pelo JavaScript.

Como é possível observar, os geradores de código foram construídos para realizar chamadas no cliente java e portanto, a aplicação web fica responsável apenas pela montagem da

lógica que guia a execução das tarefas de automação, tais como construção de repetições ou definições de valores a serem enviados junto as chamadas para que o software as realize. É possível ver um exemplo de quebra cabeças e código gerado na figura 3.17.




```
var running = true; var dias;

for (dias = 1; dias <= 31; dias += 3) {
  for (var count = 0; count < 10; count++) {
    if (running) {
    java.clickIn("../images/icons/clickBlack.png",1);
    }
    if (running) {
        running = java.type("digitar alguma coisa");
    }
    if (running) {
        java.waitImg("../images/icons/clickBlack.png", 200);
    }
  }
}
```

Figura 3.17: Exemplo de um quebra-cabeça e o código JavaScript gerado pelo Blockly para o mesmo.

4 RESULTADOS

4.1 O software automatizador

O produto final deste projeto, o software automatizador, possui uma interface sucinta, contida dentro da definição dos componentes que foram modelados para interação do usuário durante seu planejamento. O principal objetivo da quantia reduzida de elementos consiste em reduzir as interações dentro do possível considerando os requisitos e objetivos do projeto, para assim, reduzir o tempo gasto compreendendo apenas a interface do software e logo o usuário começar a estudar e compreender o quanto antes possível, como funcionam os blocos e como montar uma automação, tarefa que ainda demanda capacidade lógica. É possível ver a interface na figura 4.1.

existem, quatro botões principais na interface: a lixeira, para onde o usuário pode arrastar o quebra-cabeças para descartá-lo, um para executar a simulação que se encontra montada atualmente, um para carregar um quebra-cabeças que tenha sido salvo e outro para salvar em um arquivo XML um quebra-cabeças atualmente montado no automatizador.

Além destes quatro botões, existem os três botões na parte superior da janela, responsáveis por fechar, maximizar ou minimizar a janela. Na mesma altura, nas proximidades destes três botões, a ceta do mouse muda de símbolo, indicando que nesta região, é possível clicar e segurar para mover a janela.

Próximo aos quatro botões principais na interface ainda temos mais três botões, um círculo com um ponto dentro, que quando clicado centraliza a área de edição no quebra-cabeças atual, e dois círculos, um com o símbolo de soma e outro com o símbolo de subtração, com os quais é possível aumentar ou reduzir a visão do quebra-cabeças na área de edição, para que a visualização fique mais confortável dependendo do tamanho do quebra-cabeças.

Além dos botões descritos, temos a área de trabalho, onde a automação é montada, um espaço em branco com uma grade pontilhada, e também temos os elementos de automação armazenados em oito categorias no painel cinza a esquerda, que são:

- lógica: Blocos que permitem fazer escolhas condicionais e avaliar expressões booleanas.
- repetições: Blocos que permitem realizar repetições e *loops* baseado em contadores, variáveis ou listas.

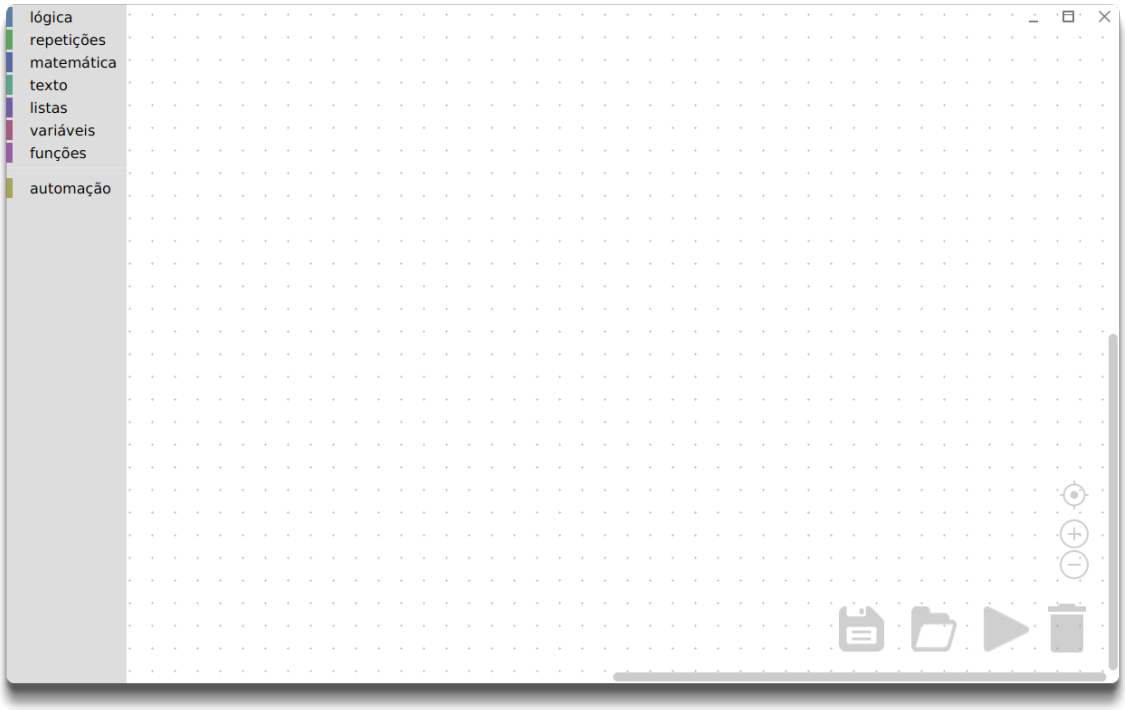


Figura 4.1: Tela inicial do software desenvolvido

- matemática: Blocos que permitem a realização de operações matemáticas, utilização de variáveis numérias e obtenção de valores numéricos aleatórios.
- texto: Blocos que permitem a criação de variáveis de texto assim como a manipulação e análise de texto, permitindo a impressão de valores em eventos ou inserimento de textos durante o fluxo de execução dos blocos.
- listas: blocos que permitem a criação, análise e manipulação de listas de valores de diversos tipos, oferecendo inclusive blocos de pesquisa e ordenação.
- variáveis: blocos que permitem criação, manipulação e utilização de variáveis em outros blocos.
- funções: Blocos que permitem a definição de funções que permitem a utilização de parâmetros e retorno de valores, incluindo inclusive a possibilidade de descrição de uma definição para a função, que descreva seu funcionamento ou utilidade.
- automações: Blocos que permitem a utilização de dispositivos de entrada padrão do sistema para manipular outros programas ou ainda observar a tela em busca da satisfação de alguma condição.

Para utilizar um dos blocos basta clicar e segurá-lo, arrastando para a área de trabalho do automatizador e soltando em uma área em branco ou com o seu encaixe alinhado com o encaixe adequado de algum outro bloco já existente na área de trabalho, para que se encaixem formando uma sequência imperativa de operações.

Das oito categorias apresentadas, a categoria “automações"contem os blocos desenvolvidos neste projeto, que permitem automatizar outros softwares. Os blocos de automação podem ser observados na figura 4.2, que, após carregados na área de trabalho do automatizador, podem ser configurados.

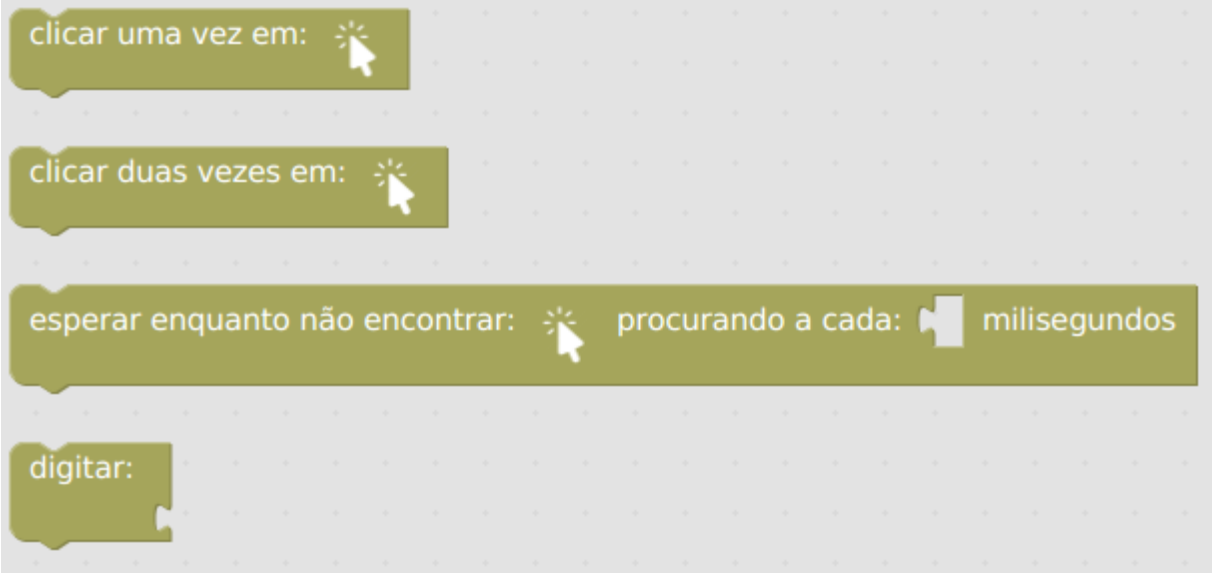


Figura 4.2: Blocos construídos para as operações de automação.

Para selecionar a imagem que representa a área onde será clicado ou pela qual será procurada na tela do computador repetidamente, o usuário deve clicar no símbolo da ceta de mouse branco que aparece no bloco. o monitor do computador será contornado por uma linha pontilhada vermelha, indicando que o mesmo pode minimizar, mover ou interagir com o sistema em busca da região da tela que queira capturar como imagem do local onde clicar ou pelo qual esperar. Assim que encontrada, o usuário pode ativar o comando de teclado ctrl+shift+c para capturar uma imagem da tela atual, posteriormente clicando, segurando e arrastando para selecionar uma área retangular indicando para o bloco onde realizar a função de clique ou procura em espera, a função de clique clicará no centro, e no caso do bloco de espera procurará por uma região igual e quando encontrada, retornara verdadeiro permitindo que a automação prossiga.

O bloco de espera procurará pela imagem na tela a cada tantos milissegundos que o

usuário pode definir inserindo um bloco do tipo número contido na categoria matemática.

Já o bloco digitar, permite a conexão de qualquer tipo de bloco que devolva um valor, tal como números e textos, que será digitado quando a automação executar este bloco. Outras formas de interação possíveis no software são, clicando e arrastando a área de trabalho para movê-la ou ainda usando o botão deletar do teclado para deletar um conjunto ou um bloco selecionado. Também é possível utilizar o comando ctrl+z para desfazer alterações como deleção de blocos.

4.2 Exemplos simples de automações

Para melhor compreender o funcionamento do software sera apresentado um quebra-cabeças simples que apresentará como o automatizador utiliza os blocos em conjunto para realizar uma tarefa, apresentando a região automatizada antes e depois da execução da automação.

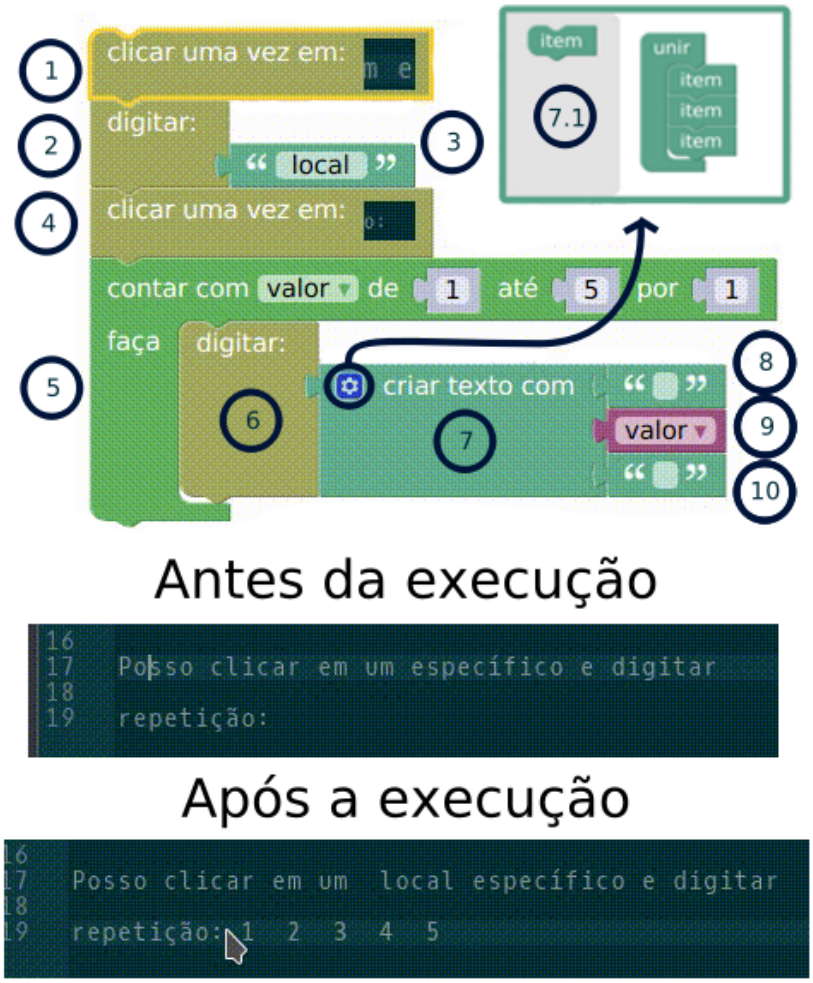


Figura 4.3: Exemplo simples de uma automação em um editor de texto.

O exemplo que pode ser visto na figura 4.3 descreve uma automatização simples de um editor de texto, onde textos são inseridos em posições específicas de um editor de texto, sendo o primeiro texto produzido diretamente pelo usuário e os seguintes, gerado pela iteração sobre uma variável, um contador, fazendo uso de blocos das categorias de automação, texto, repetições e variáveis.

Segue uma descrição de cada um dos passos realizados durante a automação do editor de texto, descrevendo a função de cada bloco seguindo a ordem enumerada na imagem 4.3:

1. Este bloco procura uma região na tela que corresponda exatamente a imagem descrita no bloco, que no caso corresponde a um espaço no editor de texto, meio de duas palavras, onde a primeira termina com “m"e a segunda começa com “e", sendo estas duas palavras “um"e “específico".

2. Em seguida ao primeiro bloco, é executado o bloco de digitação, que converte o valor do bloco acoplado a entrada deste para texto e envia o valor como dispositivo de teclado para o editor de texto, escrevendo o valor onde se encontra o cursor no editor, que é na região previamente clicada pelo automatizador.

3. Bloco que devolve um valor de texto, que o bloco anterior digitará, no caso, um bloco de criação simples de texto contendo o valor “local"mais um espaço no final.

4. Em sequência, a automação executa o segundo bloco de clique simples presente na automação, clicando na região que corresponde, como é possível ver na imagem dentro do bloco, ao fim da segunda linha 19 no editor.

5. Bloco que dá início a uma repetição, realizando as funções dos blocos dentro do mesmo a cada iteração. Este bloco itera sobre o contador “valor", começando em 1 e incrementando o valor do contador em 1 a cada repetição, até que o contador assuma o valor 5.

6. Converte o valor do bloco acoplado a entrada deste para texto e envia o valor como dispositivo de teclado para o editor de texto, escrevendo o valor onde se encontra o cursor do editor, dependendo da iteração em que se encontra.

7. Bloco que devolve um valor de texto, do tipo composto, que permite a concatenação de diversas variáveis na entrada em uma única saída de texto, que no caso desta automação, é retornado para um bloco de digitação dentro da repetição, o quinto bloco descrito. O bloco foi mudado para aceitar três entradas.

7.1. Observe que o sétimo bloco permite mutabilidade que pode ser configurada clicando na engrenagem destacada no sétimo bloco. O bloco de criação de texto composto, por padrão, apenas permite a concatenação de duas entradas. Antes de dar início a automação o bloco foi mudado, é possível adicionar quantos itens se quer como entrada no bloco de união na janela de mutação que surge ao clicar no botão de mutação do sétimo bloco, alterando assim o número de entradas do bloco de criação de texto composto.

8. Devolve um valor de texto que o bloco anterior concatenará, no início de uma variável de texto, junto com outras duas variáveis, contendo um espaço.

9. Devolve o valor armazenado no contador na iteração atual, que pode ser um valor de 1 até 5 dependendo em qual ponto da iteração a automação se encontra. O bloco anterior concatenará, no meio de uma variável de texto, junto com outras duas variáveis.

10. Devolve um valor de texto que o bloco anterior concatenará, no fim de uma variável de texto, junto com outras duas variáveis, contendo um espaço.

4.3 Validando a solução com os casos de teste

Com as funcionalidades desenvolvidas no software não é possível solucionar todas as questões observadas nos casos de teste dos 3 modelos Porem é possível solucionar em geral o problema dos modelos projetando o quebra-cabeças levando em consideração as limitações do software.

Algumas limitações principais encontradas ao modelar um quebra-cabeças para os casos de teste coletados foram:

- O software não é capaz de suavizar a criticidade com a qual procura os campos a serem clicados, logo qualquer diferença já torna impossível para o software localizar o campo novamente.

- Não é capaz de se deslocar pela interface do sistema operacional, ir para a área de trabalho ou navegar entre as janelas abertas do sistema.

- Também não é capaz de editar arquivos de texto. Embora possua diversas capacidades para processar textos dentre seus blocos, não ha nenhum bloco capaz de abrir um arquivo armazenado em alguma pasta do sistema ou ainda ler texto da tela.

- Apesar de ser capaz de esperar por mudanças gráficas na tela, o software não é capaz de tomar decisões lógicas na ocorrência de uma mudança gráfica, habilidade relevante para o tratamento de erros ou seguimento de múltiplos fluxos.

Dentre outras limitações não fundamentais, esta o fato de que o software não é capaz de pausar ou reiniciar simulações em andamento, assim como também não ha *feedback* visual do software quando o mesmo finaliza a tarefa ou encontra algum erro.

4.3.1 Simanihot

As imagens 4.4, 4.5 e 4.6 abordam a demonstração lógica em um conjunto de quebra-cabeças capaz de automatizar o modelo Simanihot para o caso de uso coletado com um conjunto hipotético de arquivos de entrada. O quebra-cabeças gerados soluciona o caso de uso dentro da lógica disponibilizada pelo conjunto de peças do software, conseguindo assim, realizar o problema principal do caso de uso, realizar diversas simulações com as variações especificadas. Algumas condições não fundamentais para a solução do caso de uso não puderam ser realizadas.

O caso de uso do modelo Simanihot define no fluxo de exceção que, em caso de qualquer interação do usuário durante a tarefa de automação, deve se pausar a simulação e esperar que o usuário recomece a automação ou cancele. Também define que no caso de encontrar algum alerta de campo preenchido incorretamente, deve encerrar a automação e enviar o usuário para as pós condição.

O automatizador não foi capaz de solucionar estas condições, devido ao fato de que não apresenta funcionalidades que permitam ao mesmo pausar, cancelar ou recomeçar uma automação, a automação após começada somente irá finalizar caso ocorra algum erro durante a automação, o automatizador seja fechado, ou de fato a automação tenha acabado.

A limitação da procura pelas regiões dos blocos de automação na tela necessita que a automação seja configurada levando em conta a necessidade de fechar e reabrir a aplicação após cada simulação, já que o automatizador não possui uma abordagem de procura pelas áreas que permita escolher as áreas a serem clicadas considerando outras condições que não a exata igualdade entre as áreas, o que leva o automatizador a falhar caso tente preencher novamente um campo que já foi alterado e portanto, produz uma diferença na imagem da região quando capturada para procurar a localização da ação de clique.

Em sequência será analisada a descrição da automação observada no conjunto de figuras 4.4, 4.5 e 4.6. Este conjunto de quebra cabeças se encontra visível em conjunto no auto-

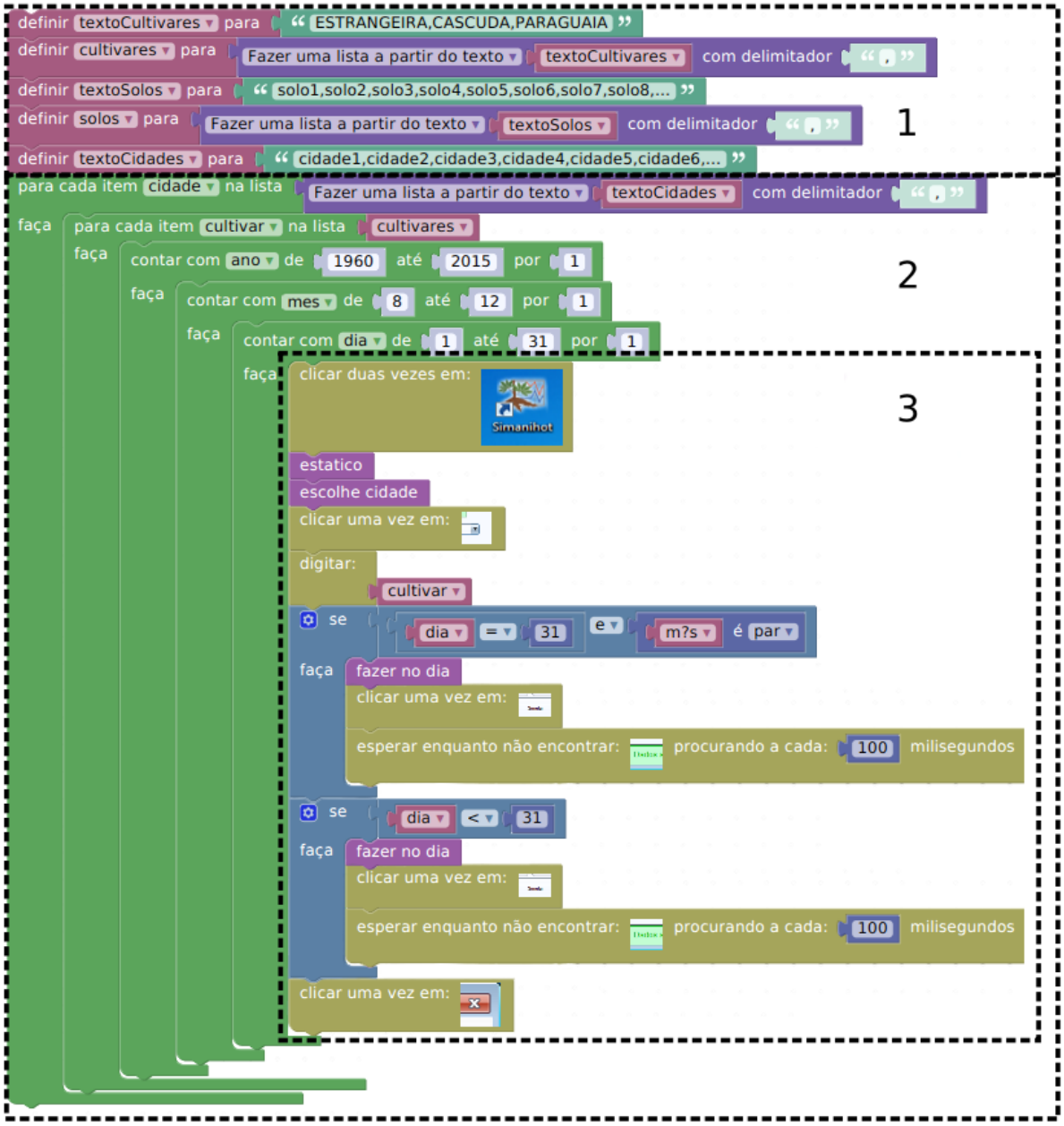


Figura 4.4: Quebra-cabeça que descreve a parte principal da automação do caso de uso do modelo Simanihot, as regiões de interesse foram demarcadas para referência durante a explanação da automação do modelo.

matizador, durante a execução, e representa uma dentre diversas formas de modelar a lógica e sequência de operações que levam a solução do caso de usos do modelo Simanihot

Analisando inicialmente a figura 4.4, foram divididos os componentes do quebra-cabeças em 3 conjuntos:

1. Este é o ponto inicial onde definimos alguns conjuntos de listas que são utilizadas durante a automação, inicialmente criamos as listas em forma de texto e posteriormente separamos os elementos por vírgula. Dentre elas temos as cultivares que serão usadas, e os

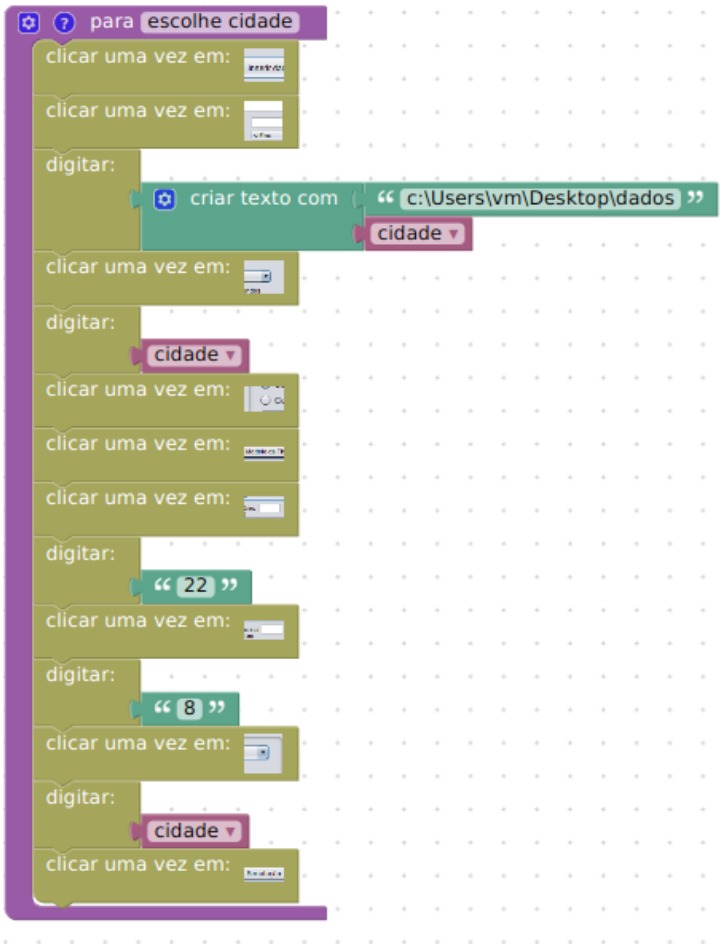


Figura 4.5: Quebra-cabeça que descreve a função que realiza escolha das cidades no modelo, chamada pelo quebra-cabeças principal apresentado na imagem 4.4.

solos. O texto contendo as cidades, por sua vez, é convertido em uma lista diretamente na repetição mais externa do segundo conjunto de blocos.

2. Neste subconjunto ocorrem as iterações necessárias para que o automatizador possa variar dentro das regras definidas nos casos de uso, as cidades, cultivares, ano mês e dia. a cada variação de dia, são rodados os blocos do terceiro subconjunto, interno ao segundo subconjunto

3. Subconjunto que realiza as operações de automação sobre o modelo. Inicialmente o modelo é executado, são chamadas as funções “estatico"descrita na figura 4.6 e “escolhe cidade"descrita na figura 4.5 e em seguida é escolhida a cultivar. Posteriormente, considerando o espaço de meses desejado para o caso de uso, é definida uma regra logica que impede a simulação de ocorrer em dias que não existam no calendário. Posteriormente caso o dia seja válido, a simulação é executada e espera-se pela validação da interface

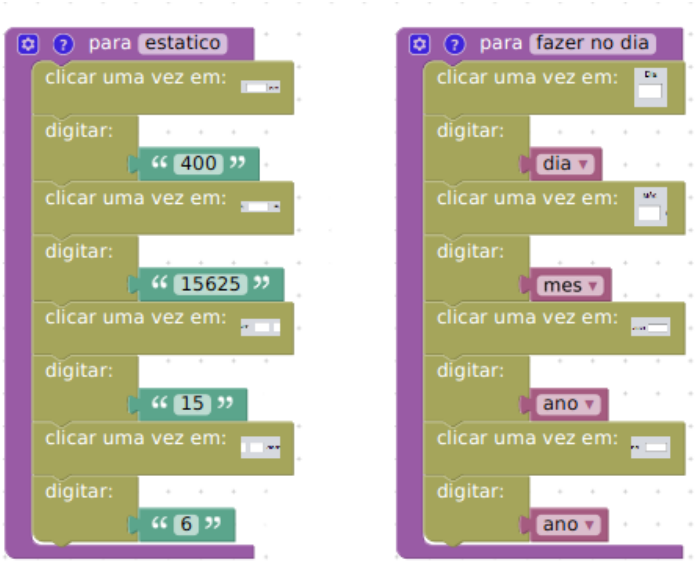


Figura 4.6: Quebra-cabeças com duas funções, “estatico"que descrevem a parte constante da automação, que não varia nas repetições e “fazer no dia"que descreve as ações que são realizadas na iteração mais interna do quebra-cabeças principal. São chamada pelo quebra-cabeças principal apresentado na imagem 4.4.

de que a simulação foi finalizada, para finalmente fechar o modelo para que possa ser executado novamente no próximo dia segundo a iteração descrita no segundo conjunto.

A função “estatico"descrita na figura 4.6 preenche os campos que não variam dentre as várias simulações que necessitam ser realizadas no caso deu uso, os campos preenchidos são concentração de CO2, densidade de plantas e o dia e mês da colheita. Na mesma imagem temos também a função “faz no dia", que preenche os campos que variam de acordo com o dia, sendo estes o dia, mês e ano do plantio, assim como o ano da colheita.

Na figura 4.5 se encontra a função “escolhe cidade", esta função se encarrega de preencher os campos que variam junto da cidade. Insere o arquivo de dados meteorológicos da cidade, construindo um caminho juntando o endereço da pasta ao nome da cidade, que é inserido no seletor de texto. Na sequência são preenchidos o local, selecionada a opção de simular com o balanço hídrico, onde é selecionado o solo correspondente a cidade atual e são definidos os valores estáticos para balanço hídrico que são a profundidade de maniva e profundidade máxima de raíz.

4.3.2 Demais casos de uso

Para o caso de uso do conjunto de modelos DSSAT, o automatizador também é capaz de realizar a automação do fluxo principal, já que o mesmo apresenta necessidades similares ao do

modelo Simanihot. Entretanto novamente não foi possível solucionar o fluxo de exceção, em especial o item 1 do fluxo de exceção do caso de uso do programa DSSAT necessita a abertura de um arquivo para avaliar os alertas de temperatura, tarefa que o automatizador não é capaz de realizar.

O software desenvolvido não foi capaz de solucionar todos os pontos fundamentais do casos de uso do modelo SoySim, o que impossibilitou a construção de uma automação que solucionasse o problema descrito no caso de uso do modelo.

O item 6 do fluxo principal do caso de uso indica que a automação deve copiar os dados a partir da linha 14 a 59 do arquivo de resultado TmpOut.txt e filtrar outros campos específicos dentro do arquivo de resultado, para posteriormente salvar todos os resultados coletados em um único arquivo de resultado. Como o software desenvolvido não é capaz de trabalhar com arquivos diretamente, o item 6 do fluxo principal do caso não pode ser resolvido.

Também não foi possível resolver o fluxo secundário e de exceções. O item 1, 2 e 3 do fluxo de exceção não pode ser resolvido pois o automatizador desenvolvido não é capaz de tomar decisões com base em eventos produzidos pelo automatizado, apenas esperar por eventos. O item 4 também não pode ser resolvido pois o automatizador desenvolvido não é capaz de pausar, cancelar ou recomeçar uma automação e o fluxo secundário não pode ser resolvido pois é dependente do item 2 no fluxo de exceção.

5 CONCLUSÃO

A proposta inicial deste trabalho consistia principalmente em apresentar uma solução de automação de interfaces gráficas de forma simplificada com base na biblioteca de programação visual Blockly dentro do contexto de modelos de simulação de culturas agrícolas. Entretanto o resultado final desta pesquisa trata-se não de uma solução total dos problemas apresentados, mas de um software que oferece a possibilidade de automatização de alguns problemas com um escopo não tão grande como o dos problemas propostos, com base nas conclusões tomadas durante o desenvolvimento, a partir dos percalços e da análise constante da amplitude da proposta e das necessidades durante a validação das abordagens em questão.

Como contribuição este trabalho apresentou uma abordagem diferenciada na programação de tarefas automatizadas expondo de forma mais simplificada lógica comum em linguagens de programação e *script* tradicionais, fornecendo uma ferramenta que pode descrever de forma muito flexível interações com a interface gráfica, permitindo em algum nível realizar a automatização de parte ou totalidade das tarefas maçantes presentes na interação com interfaces gráficas de modelos matemáticos de simulação de culturas agrícolas para os casos mais simples.

A lista a seguir, não exaustiva, expõe em ordem de relevância, uma lista de algumas melhorias que poderiam ser incorporadas em uma nova versão deste projeto:

- Tratar arquivos com conteúdo de texto, permitindo copiar arquivos para outros locais, criar arquivos com novas informações, renomeá-los e processar e extrair informações destes textos.

- Investigar possíveis exceções que podem ocorrer durante a automação, como por exemplo, a abertura indesejada de um programa, sem a solicitação do usuário e que pode bloquear a tela

- Refinar a forma com a qual o programa localiza e clica nos elementos gráficos, seja pela adição de procura por similaridades e áreas não exatas, ou por novos métodos para definição da área onde se deve clicar.

- Expandir a interação com o sistema operacional de forma que o software possa interagir enquanto procura para acessar diversas janelas ou área de trabalho, que poderiam não estar visíveis antes.

- Fornecer *feedback* visual ao finalizar automatizações ou receber erros durante a execução de uma automatização.
- Obter uma forma mais eficiente de captura de tela já que a captura de tela realizada pela classe `Robot` além de lenta pode produzir artefatos e não capturar alguns elementos dinâmicos de alguns programas.
- Oferecer blocos que simplifiquem elementos lógicos já presentes na biblioteca Blockly permitindo a formulação de quebra-cabeças de automatização mais simples.
- Refinar os detalhes em geral da aplicação como salvamento e carregamento de quebra-cabeças de forma que sejam salvas também as imagens pré selecionadas, fornecimento de configurações para troca de idioma e possibilidade de pausar, continuar e reiniciar quebra-cabeças.

REFERÊNCIAS

APPINVENTOR. Accessed:2016-02-19, http://appinventor.mit.edu/.

ARCHANJO, G.; ANDRIJAUSKAS, F.; MUÑOZ, D. Marvin–A Tool for Image Processing Algorithm Development. **Technical Posters Proceedings of XXI Brazilian Symposium of Computer Graphics and Image Processing**, [S.l.], v.1, n.1, p.5–6, 2008.

ATK. Accessed: 2016-06-18, https://developer.gnome.org/atk/.

AUTOIT. Accessed: 2016-02-19, https://www.autoitscript.com/site/autoit/.

AUTOMATOR. Accessed: 2016-02-19, https://developer.apple.com/library/mac/documentation/AppleApplications/Conceptual/AutomatorConcepts/Automator.html.

BARRY, E. J.; KEMERER, C. F.; SLAUGHTER, S. A. How software process automation affects software evolution: a longitudinal empirical analysis. **JOURNAL OF SOFTWARE MAINTENANCE AND EVOLUTION: RESEARCH AND PRAC**, [S.l.], p.31, 2007.

BLOCKLYGAMES. Accessed: 2016-02-19, https://blockly-games.appspot.com/.

BLOCKLYRESOURCE. Accessed: 2016-02-19, https://developers.google.com/blocklyResource/.

DECISION Support System for Agrotechnology Transfer (DSSAT). Accessed: 2016-02-27, http://dssat.net/.

DONMEZ, M. A.; SOONS, J. A. **Impacts of Automation on Precision**. [S.l.: s.n.], 2008. p.117–126.

GNU Make. Accessed: 2016-02-19, https://www.gnu.org/software/make/.

KOEGEL, J.; HEINES, J. M. Improving Visual Programming Languages for Multimedia Authoring. **World Conference on Educational Multimedia and Hypermedia**, [S.l.], p.8, 1993.

MARRON, A.; WEISS, G.; WIENER, G. A Decentralized Approach for Programming Interactive Applications with JavaScript and Blockly. **AGERE!**, [S.l.], p.13, 2012.

MYERS, G. J.; SANDLER, C.; BADGETT, T. **The art of software testing**. [S.l.]: John Wiley & Sons, 2011.

NOF, S. Y. **Springer Handbook of Automation**. [S.l.]: Springer, 2009.

OPENCV. Accessed: 2016-06-03, http://opencv.org/.

PARASURAMAN, R.; SHERIDAN, T. B.; WICKENS, C. D. A Model for Types and Levels of Human Interaction with Automation. **IEEE Systems, Man, and Cybernetics Society**, [S.l.], v.1, n.1, p.286–297, 2000.

ROBOT Framework. Accessed: 2016-01-12, http://robotframework.org/.

SHU, N. C. **Visual Programming**. [S.l.]: Van Nostrand Reinhold, 1988.

SIKULI. Accessed: 2016-04-19, http://www.sikuli.org/.

SILVEIRA JúNIOR, G. da et al. **Análise da ferramenta de programação visual blockly como recurso educacional no ensino de programação**. [S.l.: s.n.], 2015.

SIMANIHOT. Accessed:2016-02-27, http://w3.ufsm.br/simanihot/.

SOYSIM. Accessed: 2016-02-27, http://soysim.unl.edu/.

STRECK, N. A. et al. Simanihot: um modelo de simulação do crescimento, desenvolvimento e produtividade de mandioca. **13ž Reunião Técnica Estadual da mandioca e 5ž Reunião Técnica Estadual da batata-doce,Cerro Largo -RS: Emater-RS**, [S.l.], v.1, n.1, p.55, 2015.

TESTCOMPLETE. Accessed: 2016-04-19, https://smartbear.com/product/testcomplete/overview/.

UIPATH. Accessed: 2016-04-19, http://www.uipath.com/.

UNIX Shell. Accessed: 2016-02-19, http://www.cs.columbia.edu/~sauce/tutorial/shell.html.

VENKATRAMAN, N. IT-Enabled Business Transformation: from automation to business scope redefinition. **ABI/INFORM Global**, [S.l.], p.73, 1994.