PONTIFÍCIA UNIVERSIDADE CATÓLICA DE SÃO PAULO

EMERSON TERUHIKO WATANABE

FERRAMENTA DE PROTOTIPAGEM COM LÓGICA DE PROGRAMAÇÃO COMO FACILITADOR NO DESENVOLVIMENTO DE SISTEMAS

ARQUITETURA CONCEITUAL

São Paulo

2016

PONTIFÍCIA UNIVERSIDADE CATÓLICA DE SÃO PAULO

EMERSON TERUHIKO WATANABE

FERRAMENTA DE PROTOTIPAGEM COM LÓGICA DE PROGRAMAÇÃO COMO FACILITADOR NO DESENVOLVIMENTO DE SISTEMAS

ARQUITETURA CONCEITUAL

Monografia apresentada ao Curso de Especialização em Engenharia de Software da Pontifícia Universidade Católica de São Paulo, como requisito parcial para obtenção do título de Especialista em Engenharia de Software, orientado pelo Prof. Msc. André Luiz Garcia Pereira.

São Paulo

2016

Dedico este trabalho a minha esposa Flávia Moreira que me apoia e incentiva em todos os momentos.

AGRADECIMENTOS

Agradeço aos meus pais por todo o amor, apoio, paciência e compreensão que tiveram comigo nesta jornada.

Aos professores que me ensinaram e incentivaram para que eu não desistisse e continuasse a estudar. Em especial fico eternamente grato ao professor e orientador André Luiz Garcia Pereira, que teve a paciência para me ensinar e orientar mesmo em um período difícil da sua vida.

RESUMO

Há diversas ferramentas no mercado utilizadas para prototipagem de telas de softwares, entretanto, estas são estáticas, sem atualizações do conteúdo e sem regras de navegações entre as telas, portanto não podem ser utilizados como um software completo. Ao mesmo tempo, existem ferramentas utilizadas por programadores de software que possuem a funcionalidade de prototipagem, porém, estas são complexa se necessitam de conhecimento técnico em programação. Por outro lado, há plataformas como Scratch e Alice, que são utilizadas por crianças e adolescentes para criar jogos de forma lúdica com o uso da lógica de programação para dar vida aos elementos dos jogos. Assim, este trabalho tem como objetivo criar a arquitetura de uma ferramenta que possua os conceitos das ferramentas de desenvolvimento e do Scratch e Alice para possibilitar que as pessoas com pouco conhecimento técnico em programação consigam criar seus próprios softwares.

Palavras chaves: VPE, Arquitetura, VPL, Scratch, Alice.

ABSTRACT

There are several tools on the market used for prototyping software screens, however, these are static, no updates of content and no rules of navigations between the screens, therefore cannot be used as a full-featured software. At the same time, there are tools used by software developers who own prototyping functionality, but, these are complex and require technical knowledge in programming. On the other hand, there are platforms such as Scratch and Alice, which are used by children and teenagers to create playful shape games using the programming logic to give life to the elements of the games. Thus, this work aims to create the architecture of a tool that has the concepts of development tools and Scratch and Alice to enable people with little technical knowledge in programming can create their own software.

Keywords: VPE, Architecture, VPL, Scratch, Alice.

LISTA DE ILUSTRAÇÕES

Figura 1 – Exemplo de VPE - Microsoft Visual Studio (MALIK, 2015) 10
Figura 2 – Drag and drop de componentes de tela (PENDSE, 2009) 11
Figura 3 – Exemplo de um programa criado para no Lego Mindstorms 12
Figura 4 – Exemplo de um robô do kit Lego Mindstorm (CONSTRUCTING... 2016) 13
Figura 5 – Programa Scratch 14
Figura 6 – Ferramenta Alice (ALICE... 2016) 15
Figura 7 – Galeria de objetos (BARROS et al., 2012) 16
Figura 8 – Arquitetura básica 18
Figura 9 – Modelo de domínio 24

SUMÁRIO

1 INTRODUÇÃO 7
2 CONCEITOS GERAIS 9
2.1 Visual Programming Environment (VPE) 9
2.2 Visual Programming Language (VPL) 11
2.2.1 VPL conhecidas 13
3 ARQUITETURA DA FERRAMENTA PARA CRIAÇÃO DE UM SISTEMA 17
3.1 Motivação para o desenvolvimento desta ferramenta 17
3.2 Arquitetura da ferramenta 18
3.3 Apoio ao desenvolvimento 22
4 VANTAGENS E DESVANTAGENS DA FERRAMENTA PROPOSTA 25
4.1 Vantagens 25
4.2 Desvantagens 26
5 CONCLUSÃO 27
6 PESQUISA BIBLIOGRÁFICA 29
6.1 Livros e artigos: 29
6.2 Web Referências 30

1 INTRODUÇÃO

Há um grande número de ferramentas de prototipagem no mercado, a qual se propõe a disponibilizar um ambiente para desenhar ou criar telas de sistemas por meio de modelos de componentes gráficos pré-definidos, porém, se restringe a telas de sistemas estáticas, sem funcionalidade ou comportamento. Essas ferramentas, podem ser utilizadas por pessoas sem conhecimento específico na área de tecnologia da informação (TI).

Embora, há diversas ferramentas, como a Integrated Development Environment (IDE) para desenvolvimento de sistemas, as quais possuem funcionalidades de prototipagem, mas não são ideais para pessoas sem conhecimento de linguagem de programação, sendo uma ferramenta complexa e utilizada por profissionais da área de tecnologia.

Há também no âmbito educacional ferramentas criadas para crianças aprenderem lógica de programação por meio da utilização de blocos representados por desenhos de quebra-cabeça que ao serem interligados ou montados em ordem, estruturam blocos lógicos da programação convencional, um exemplo de software com esta característica é o Scratch o qual foi desenvolvido pelo *Massachusetts Institute of Technology* (*MIT*).

Com isso, embasado nos conceitos de prototipagem, nas funcionalidades das IDE e nas características do Scratch, é possível criar uma ferramenta que possibilite a criação de sistemas menos complexos de um modo prático, cujo público são pessoas e empresas que querem criar sistemas, mas não possuem conhecimento técnico de programação suficiente.

Este trabalho tem o objetivo de demonstrar que a criação de sistemas pode ser feita por pessoas sem muito conhecimento técnico em programação, utilizando-se de uma ferramenta contendo uma área de trabalho para geração de protótipos de telas, e uma área de trabalho para geração da lógica de programação por meio da utilização de blocos lógicos representados por ícones. Para um melhor entendimento do conceito, será também apresentado a arquitetura de uma ferramenta de forma a possibilitar que este seja um dos precursores de ferramentas mais complexas, mas cumpra com a proposta de ser utilizado por pessoas com pouco conhecimento em programação.

Como resultado, demonstrasse que os conceitos de prototipagem de tela e lógica de programação podem ser utilizados em conjunto para facilitar o desenvolvimento de sistemas.

Neste trabalho serão produzidos:

- Modelo da arquitetura da ferramenta Visual Programming Language (VPL);
- Diagramas de domínio para possibilitar o entendimento conceitual da solução.

Espera-se que este trabalho seja utilizado como base para melhorar as ferramentas já existentes e que novos conceitos sejam incorporados. Além disso, será importante se a ferramenta for implementada de forma completa com todas as condições lógicas e acrescentado componentes de apoio ao desenvolvimento.

No capítulo 1, “Revisão Bibliográfica”, são abordados trabalhos recentes sobre:

- Características da Visual Programming Environment (VPE);
- Características da VPL;
- Sistemas que utilizam blocos lógicos visuais que são utilizados para criação de jogos e animações.

No capítulo 2, “Arquitetura da ferramenta para criação de um sistema”, são descritos o desenvolvimento das seguintes atividades:

- Descrição da motivação e das principais funcionalidades que a ferramenta deve possuir;
- Criação do modelo da arquitetura da ferramenta;
- Criação do modelo de domínio da ferramenta.

No capítulo 3, “Vantagens e desvantagens da ferramenta proposta”, será descrito algumas das vantagens e desvantagens de utilizar a ferramenta.

No capítulo 4, ”Conclusão”, é feita uma síntese dos pontos abordados e implementados, além de sugestões de melhorias para futuros estudos.

2 CONCEITOS GERAIS

2.1 Visual Programming Environment (VPE)

A linguagem de programação evoluiu durante o tempo e ficou cada vez mais complexa e difícil de se manter em termos de criação e manutenção. O grande número de softwares desenvolvidos para resolver diversos tipos de problemas do dia a dia das pessoas, a quantidade crescente da demanda de desenvolvimento de novos softwares e a necessidade de se criar mais com menos tempo foram fatores que levaram a indústria de software a se preparar melhor para ter mais produtividade e qualidade, assim, surgiram as ferramentas de desenvolvimento, conhecidas como IDE (Integrated Development Environment) ou VPE (Visual Programming Environment) (AMBIENTE... 2016).

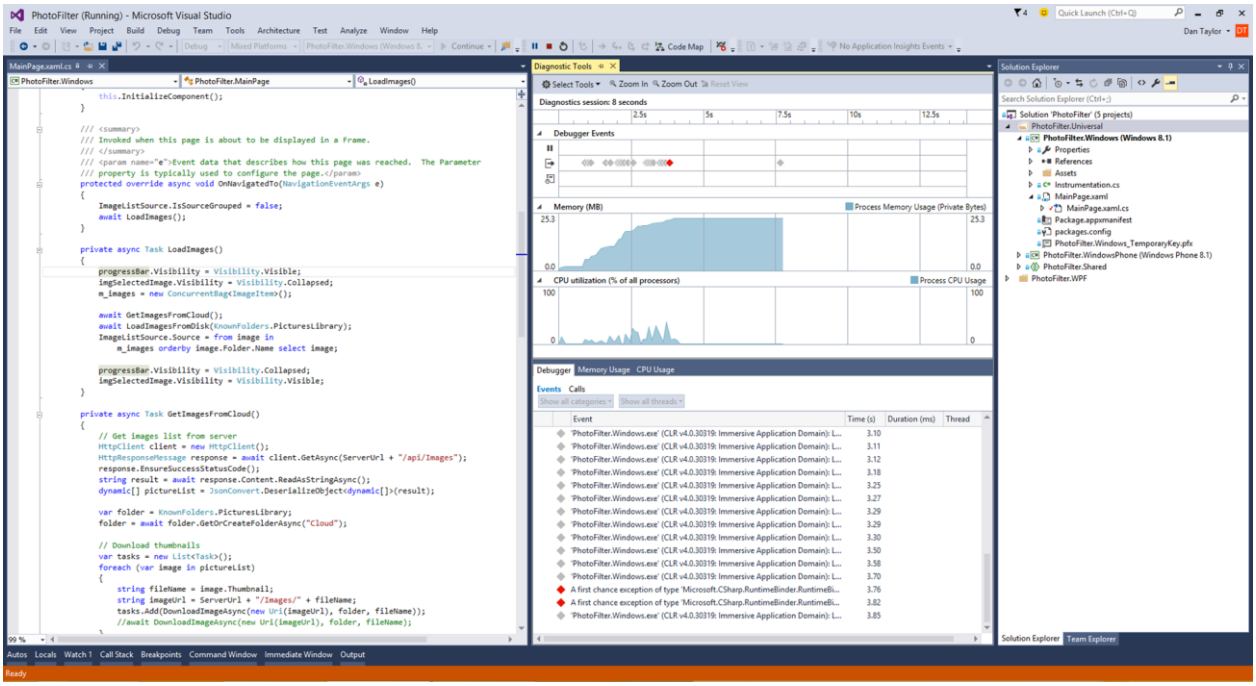
As VPE possuem diversas características, porém, a mais importante é a integração entre diversas funções necessárias para o desenvolvedor criar um programa. As características e funções existentes na maioria são: Editor de código fonte, compilador, depurador, modelagem, deploy e teste automatizado (INTEGRATED... 2016).

Estas VPE são ferramentas que auxiliam o desenvolvedor na produtividade, e esta característica ficou conhecida como *Rapid Application Development* (RAD), (RAPID... 2016). Ferramentas como Eclipse, Rational Developer Studio, JDeveloper, NetBeans, Delphi, Visual Studio entre outros, são extremamente úteis para facilitar nas atividades mais simples como editar o código, compilar e gerar o software final (CORDEIRO, 2013).

Por outro lado, apesar de aceito pela comunidade de desenvolvedores que as VPE ajudam na questão de produtividade, há uma pesquisa de Zayour e Hajjdiab (2013) que contesta a sensação de produtividade, já que, segundo eles, apesar das IDE ajudarem em várias tarefas, isto produz um efeito contrário na questão de implementação/debug, pois, o desenvolvedor utiliza de forma excessiva a funcionalidade de debug, o que lhe faz perder muito tempo para encontrar problemas e resolvê-los, assim, deve-se planejar e modelar os sistemas antes da implementação para que o ganho de produtividade com a ferramenta seja aproveitado ao máximo.

Empresas que desenvolvem as IDE (DICAS... [2015]) citam o uso do *IntelliSense* (INTELLIGENT... 2016), os atalhos para realizar *Code Snippets, Outline* para agrupar trechos de códigos, *drag and drop* de componentes, criação de testes unitários automáticos (Avery, 2005), etc., como maneiras de agilizar o desenvolvimento de um software. A Figura 1 é um exemplo de uma VPE utilizada comercialmente.

Figura 1 – Exemplo de VPE - Microsoft Visual Studio (MALIK, 2015)

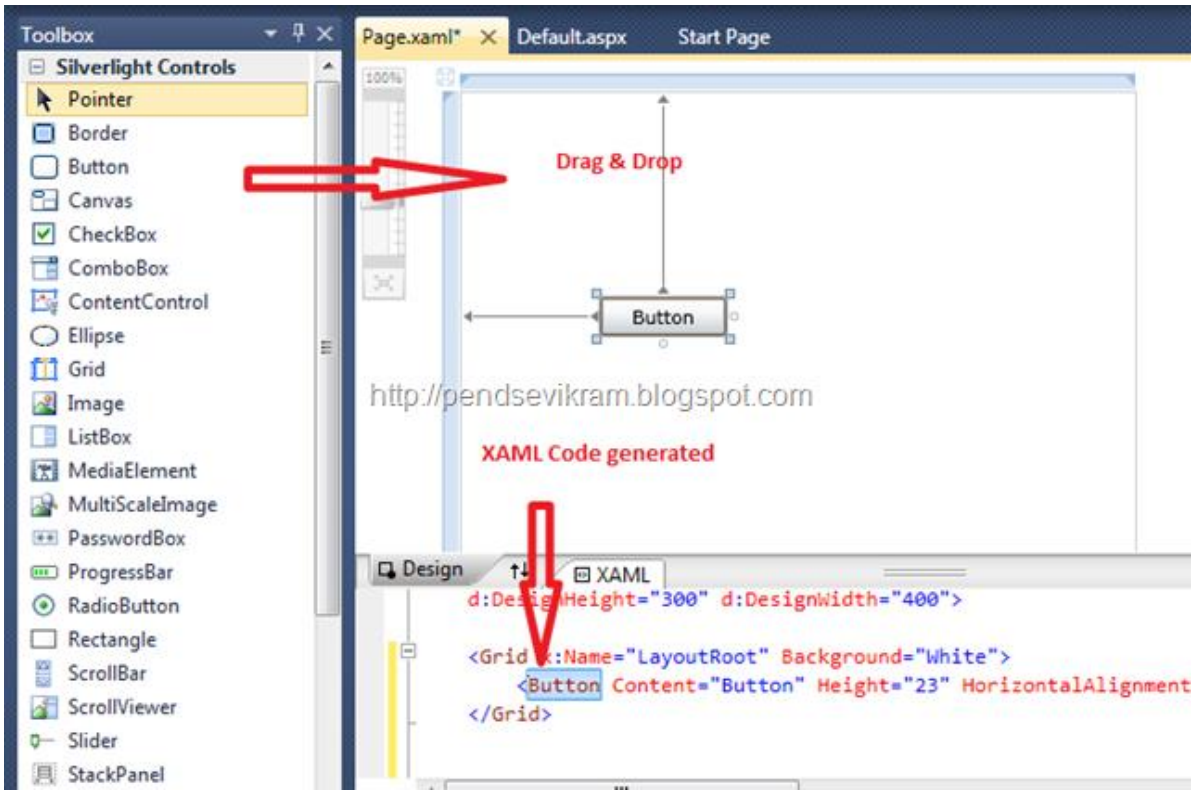


Dentre todas as funcionalidades disponíveis na IDE, uma das mais utilizada é o *drag and drop* de componentes de telas para criação de formulários, a qual será utilizada para a interação do usuário final, como pode ser visto na Figura 2. Esta funcionalidade não é utilizada somente nas IDE, está presente em muitos programas gráficos como o Adobe Photoshop, Corel Draw e Autodesk 3ds Max, além dos programas de edição de textos como o Microsoft Word, Excel e similares.

Na IDE, o *drag and drop* é muito utilizado para facilitar no reuso dos componentes de telas, pois estes componentes possuem propriedades padrões definidas em código, que podem ser alteradas conforme necessidade. Isto é muito prático, já que o desenvolvedor não precisa implementar todo o código necessário

para criar os componentes e definir as propriedades no código. Há muitos programadores mais experientes que não utilizam o recurso de *drag and drop* disponível pelas IDE, e preferem criar os objetos dinamicamente via linha de código, pois, na maioria dos casos eles querem personalizar os componentes de forma que atendam às necessidades.

Figura 2 – Drag and drop de componentes de tela (PENDSE, 2009)



2.2 Visual Programming Language (VPL)

A linguagem de programação visual conhecida pela sigla em inglês VPL (Visual Programming Language) foi concebida utilizando o conceito de fácil utilização e manipulação.

Em 1967, Seymour Papert, Daniel G. Bobrow, Wally Feurzeig e Cynthia Solomon iniciaram uma pesquisa no Massachusetts Institute of Technology (MIT) que levou na construção do robô LOGO Turtle (LOGO... 2016), tal qual se movimentava com comandos enviados pelo programa VPL que foi batizado como

LOGO, este, utilizava comandos básicos como “move 10 forward”, “turn up”, “move 10 forward” (CONSTRUCTING... 2016). O projeto do LOGO foi desenvolvido para ser utilizado na área educacional e desta forma foi concebido para dar apoio a teoria chamada de Construcionismo, proposta Papert, este que foi baseada na teoria do Construtivismo de Piaget, porém com a diferença de que, Papert, acreditava que era necessário o experimentalismo para que o aprendizado fosse eficaz, desta forma, era possível aprender experimentando, e para isso ele cita que o uso de ferramentas e recursos computacionais são importantes para o aprendizado (CONSTRUCTIONISM... 2016).

Após a publicação do seu livro *Mindstorms – Children, Computers and Powerful Ideas*, Papert, fez parceria com a empresa Lego para a criação do Lego Mindstorms (LEGO... 2016), para ser utilizada como recurso pedagógico utilizando o computador, devido ao seu fácil manuseio para criação de robôs com as tradicionais peças do Lego, e aliado a um programa VPL criado pela LabView para desenvolver programas para o Lego Mindstorms, veja um exemplo de programa na Figura 3, e na Figura 4 podemos um exemplo de um robô montado com peças do Kit Lego Mindstorm e controlado pela VPL da LabView.

Figura 3 – Exemplo de um programa criado para no Lego Mindstorms

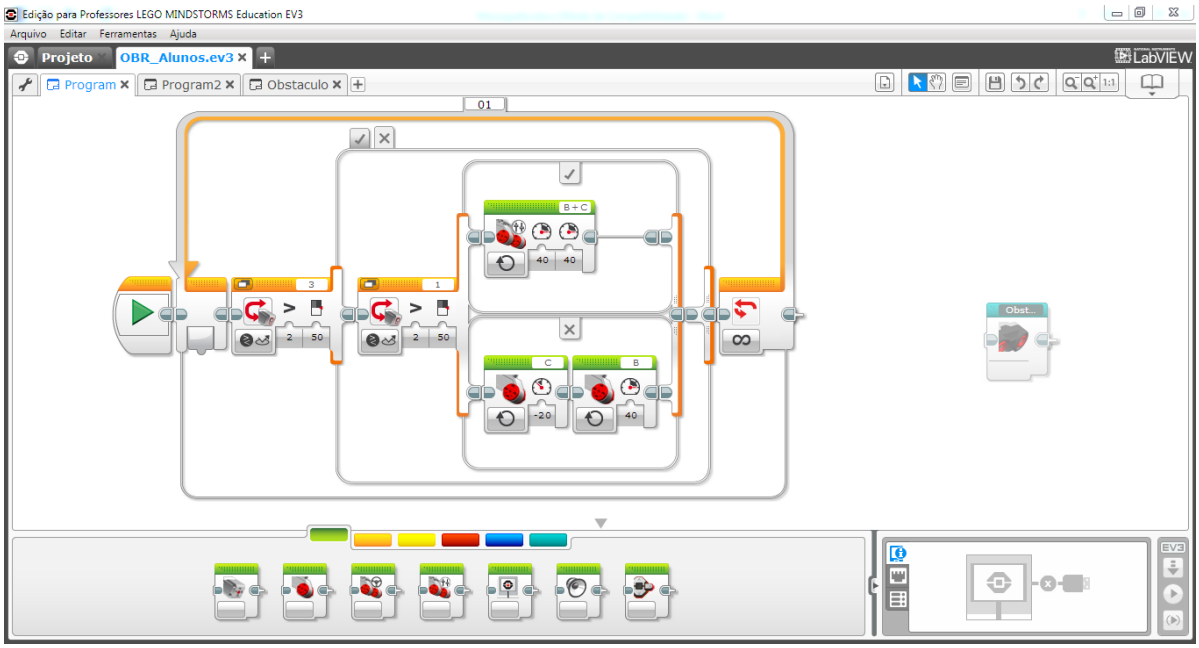
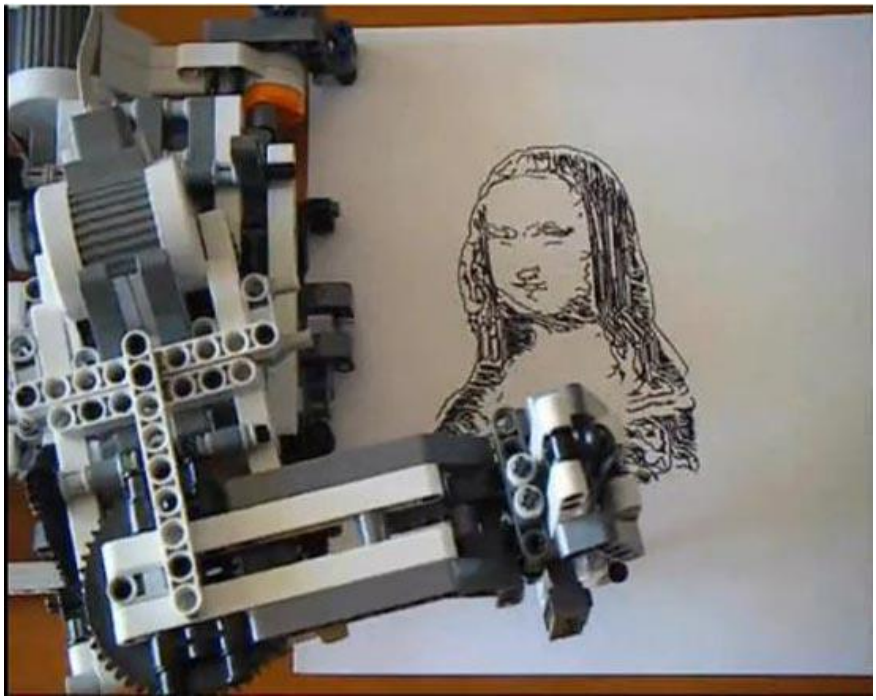


Figura 4 – Exemplo de um robô do kit Lego Mindstorm (CONSTRUCTING... 2016)



2.2.1 VPL conhecidas

Após o projeto LOGO outras VPL foram criadas, das quais as mais conhecidas são, Scratch, Alice, Blockly e App Inventor. Neste trabalho, detalha-se o Scratch e Alice, já que os mesmos são suficientes para entendimento e exemplos de como funcionam as VPL.

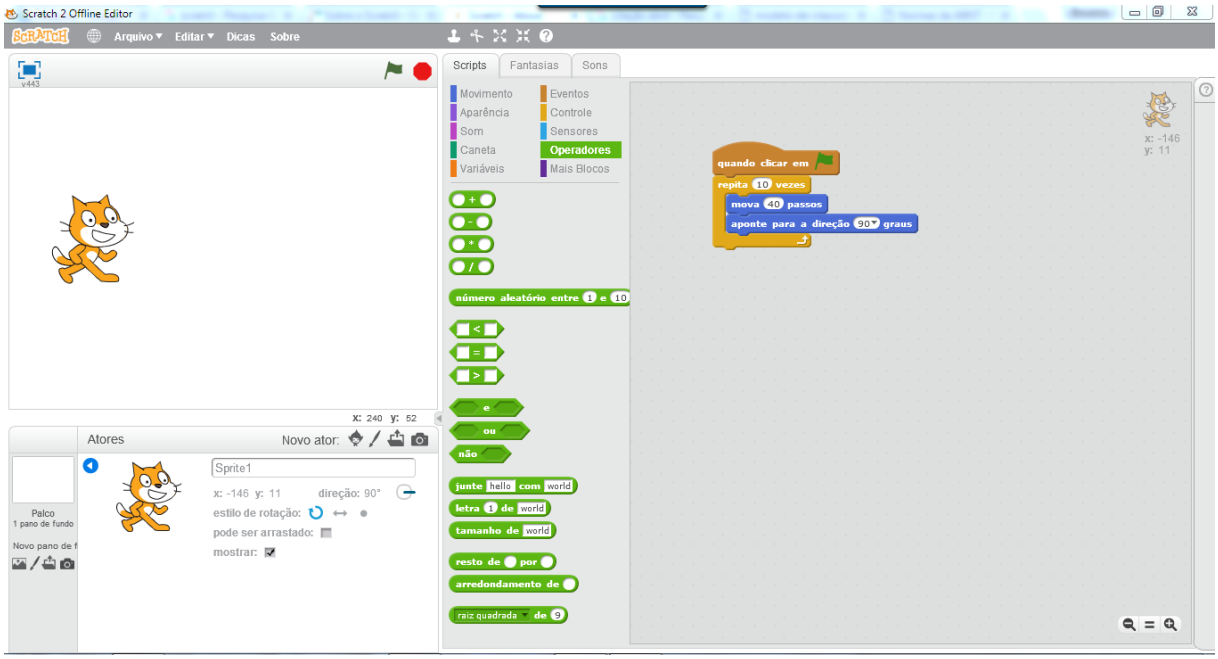
2.2.1.1 Scratch

Umas das VPL mais conhecidas é o Scratch, de acordo com o site do Scratch Fondation ele foi desenvolvido por Mitch Resnick em 2003 no MIT (SCRATCH... 2016), utiliza blocos lógicos em forma de quebra cabeça, possui recursos de sons, animações, jogos e outros. É possível baixar o Scratch gratuitamente, no site também existe espaço para compartilhamento e fórum de discussões.

> O Scratch está concebido especialmente para jovens entre os 8 e os 16 anos de idade, mas é usado por pessoas com todas as idades. Milhões de pessoas criam projectos Scratch numa grande variedade de contextos, incluindo lares, escolas, museus, bibliotecas e centros comunitários. (ABOUT... 2016)

Na Figura 5 podemos ver um exemplo do Scratch em uso, a qual visualiza-se no canto direito um espaço para incluir blocos semelhantes a um quebra cabeça, a qual representam os blocos lógicos da linguagem de programação, neste espaço é onde desenvolve-se todo o algoritmo para o funcionamento do jogo ou animação desejado. No meio há a área dos blocos lógicos disponíveis para a área mencionada anteriormente. Na área do canto superior esquerdo pode-se notar um gato, símbolo do Scratch, que é o personagem principal e apresentado inicialmente ao abrir o programa, esta área é onde visualiza-se a execução do programa pré-criado na área dos blocos lógicos. Na parte esquerda abaixo da área onde está o gato, está a área com a lista de personagens que poderão ser incluídos na área de execução.

Figura 5 – Programa Scratch

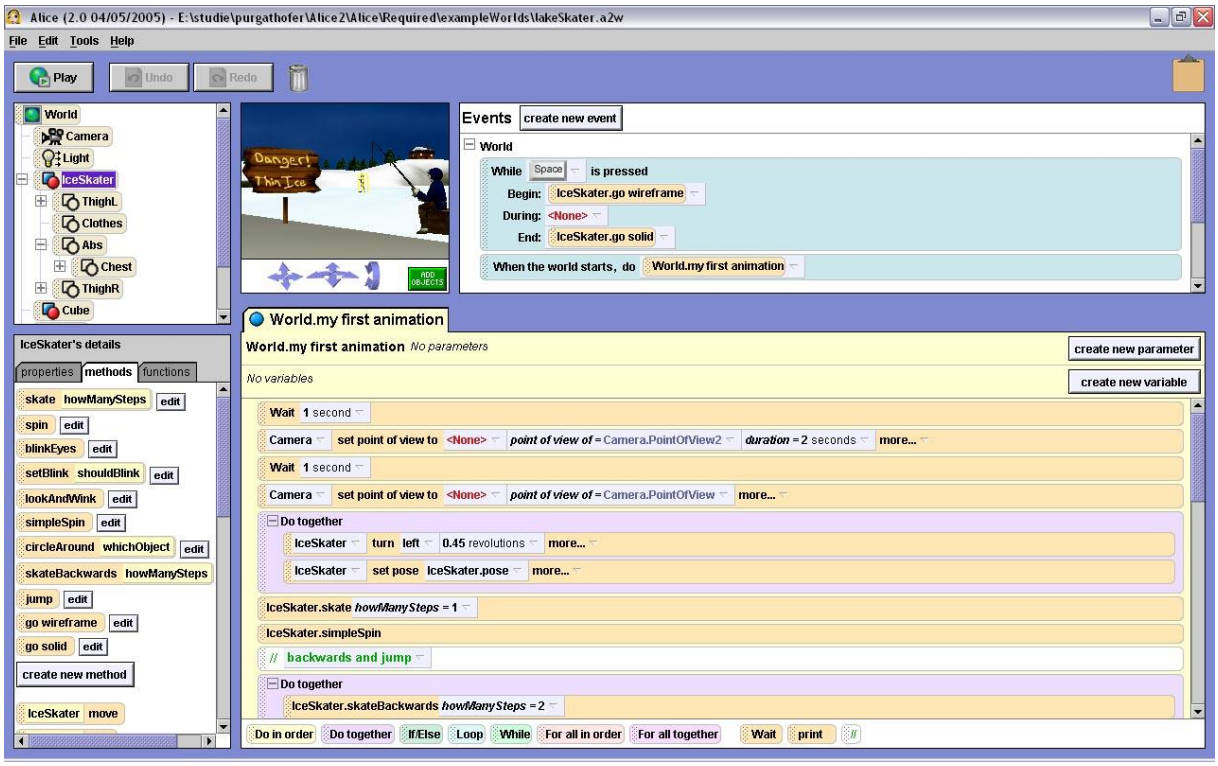


2.2.1.2 Alice

Criado por Carnegie Mellon University, Alice (2016), assim como o Scratch, também é uma ferramenta VPL e é utilizado principalmente por instituições que ensinam programação para jovens e adultos. Esta VPL visa facilitar o ensino da programação orientado a objetos utilizando-se de um ambiente que contém interface para utilizar objetos 3D, a qual propicia a criação de jogos e animações iterativos. As

animações são feitas utilizando-se instruções semelhante à de uma linguagem de programação como C++, Java e C#, porém a lógica do programa é feita arrastando-se blocos lógicos em forma gráfica, semelhante ao Scratch que utiliza peças em forma de quebra-cabeça. Com isso, o Alice se torna uma ferramenta que facilita o entendimento da lógica de programação com a manipulação de objetos por meio da criação de animações em um mundo virtual criado pelo aluno. A Figura 6 mostra o ambiente da ferramenta Alice.

Figura 6 – Ferramenta Alice (ALICE... 2016)



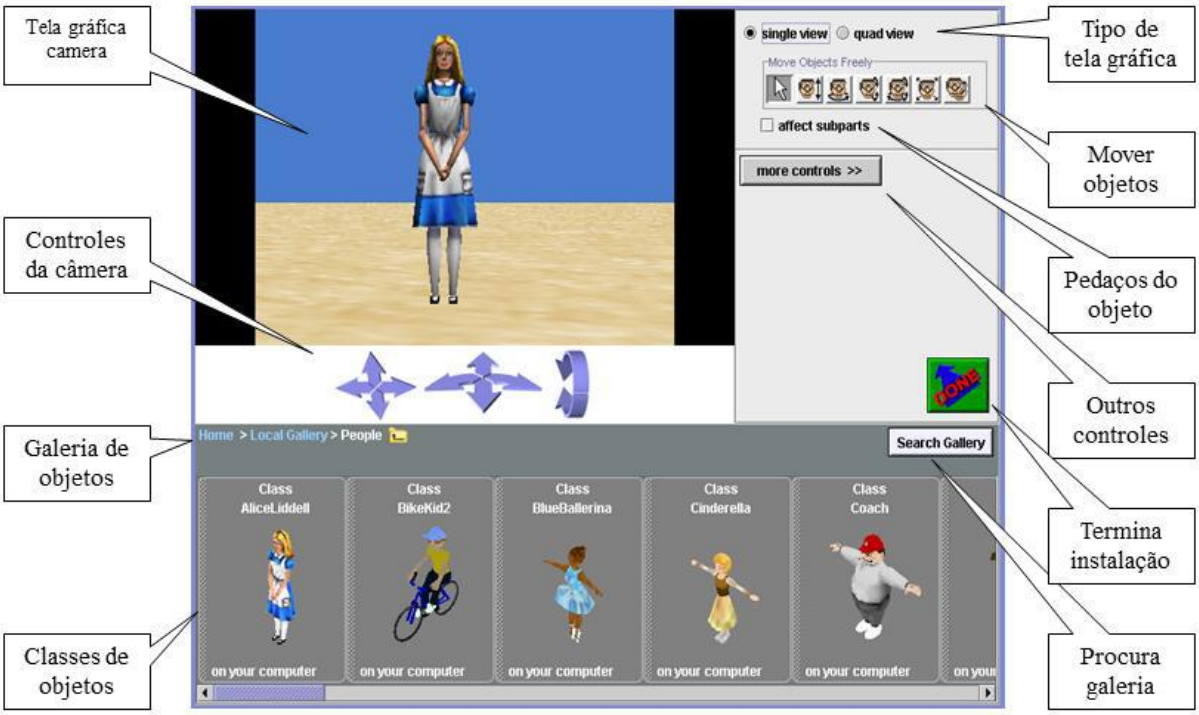
Na Figura 6 nota-se algumas semelhanças no ambiente do Alice com as IDE como Visual Studio, Eclipse e Delphi. Como mencionado abaixo:

> O que se chama nos ambientes de programação de PROJETO no ALICE é referenciado como MUNDO (WORLD). Há uma Árvore de Objetos presente no ambiente, como ocorre em outros ambientes de programação. Cada objeto instanciado no Mundo (Projeto) possui detalhamentos que podem ser observados em área específica (Área de detalhes), o que em outras linguagens de programação seria referenciado como Inspetor de Objetos. Nessa Área de detalhes há, para cada objeto, informações relacionadas às suas Propriedades, Métodos e Funções. (BARROS et al., 2012)

Na área de codificação verifica-se que as linhas de código estão dispostas em formato de blocos onde pode-se notar os comandos de chamada dos métodos e de

blocos lógicos aninhados de forma que possam ficar legíveis. Nota-se também que a disposição dos blocos e a forma como são incluídos não deixam o desenvolvedor errar a sintaxe, o que é uma grande vantagem no âmbito do ensino da programação, pois, uma das dificuldades apresentadas pelos alunos é o erro de sintaxes (BARROS et al., 2012).

Figura 7 – Galeria de objetos (BARROS et al., 2012)



Na Figura 7 vê-se a apresentação dos objetos disponíveis para incluir na cena. Os objetos são divididos por grupos, cada qual possui suas características como veículos, pessoas, prédios, etc. Após a inclusão do objeto na tela é possível movimenta-los para colocá-los no local inicial.

Ao analisar as Figuras 6 e 7 é nítido que o uso do recurso de *drag and dro*p é primordial para esta ferramenta, pois é utilizado tanto na área de codificação como na área da animação.

3 ARQUITETURA DA FERRAMENTA PARA CRIAÇÃO DE UM SISTEMA

3.1 Motivação para o desenvolvimento desta ferramenta

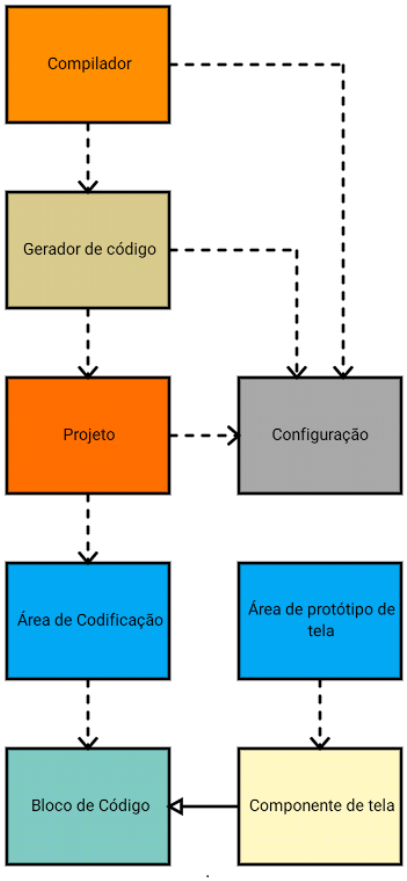
A programação está cada vez mais evoluída, e chegará um dia que não precisaremos mais tratar todos os métodos existentes dentro de um programa. Isto porque as abstrações do funcionamento dos componentes inerentes ao desenvolvimento serão tratadas internamente, de forma transparente e sem preocupação por parte dos programadores.

Com o tempo, um dia será possível apenas falar para o sistema qual a nossa necessidade e o mesmo executarão várias tarefas sem que precisemos interagir o tempo todo com eles, ou seja, o princípio é fazer com que os sistemas sejam independentes e abstratos, e então, tudo que nos preocupamos na programação, como alocação de memória, fechamento de banco de dados, timeouts, exceptions, etc, não serão problemas, pois, estarão totalmente consolidadas e confiáveis.

A VPL, pode ser o início desta evolução da programação, de uma forma tão abrangente, que será aplicada a usuários sem tanto conhecimento em programação. O ideal é que qualquer pessoa fosse capaz de criar um programa somente escrevendo suas ideias ou desenhando o seu fluxo de trabalho. Para se chegar neste patamar a própria maneira de interação com o computador deve mudar, ou seja, a interface gráfica que temos hoje seria muito diferente, de forma, que os usuários manipulassem os objetos como manipulamos coisas no mundo real, pegando, arrastando, com o intuito de cada coisa fazer sentido e executar uma tarefa por si só. Partindo deste ideal, temos que a primeira mudança se dá no âmbito da componentização completa e fácil para montar e delegar. A confiabilidade seria inerente ao restante dos componentes que já teriam outras funcionalidades. O início deste processo de evolução seria a dolorosa e demorada, porém, com o tempo a colaboração e o compartilhamento dos componentes desenvolvidos evoluiriam naturalmente.

3.2 Arquitetura da ferramenta

Figura 8 – Arquitetura básica



Na Figura 8 pode-se ver a arquitetura básica da ferramenta VPL, no mesmo identificasse os componentes Compilador, Gerador de Código, projeto, Configuração, área de Codificação, área de protótipo de tela, blocos de Código e Componentes de tela.

- Compilador: O Compilador é o componente responsável por gerar o código binário final, que será utilizado para o usuário final, tal como executável, biblioteca, etc. Este componente tem como entrada os arquivos gerados pelo Gerador de Código e as configurações do componente de Configuração. O mesmo será executado quando operador da ferramenta escolher a opção de compilação do código que foi gerado anteriormente. Diante disso, o compilador só executará a compilação mediante a existência de um código gerado.

Importante ressaltar que este componente fará uso dos compiladores previamente instalados, conforme linguagem escolhida, por exemplo, caso desejado que seja feita a compilação de um código Java, então, o operador deverá ter instalado todos os requisitos desta plataforma, igualmente se escolhida a linguagem C# ou C++, então, deve-se instalar os componentes necessários para tal. Para tanto, o compilador fará uso das configurações existentes no componente de configuração presente na ferramenta, tal qual está descrito nos itens abaixo.

- Gerador de código: Este componente é o responsável por gerar insumo para o compilador. É responsável por ler o código VPL, armazenado no componente Projeto, e traduzir no código desejado. O componente Configuração contém a informação de qual é o código de saída (Java, C#, C++, etc.). Para cada linguagem configurada, este componente deverá ter uma extensão a qual lhe dará condições de gerar o código nesta linguagem. Para tanto, este componente contém uma interface a qual a entrada para leitura da linguagem VPL será conhecida e imutável, mas a saída dependerá de cada extensão desenvolvida.
- Projeto: Este componente contém toda a estrutura criada no projeto VPL, como as configurações escolhidas para o projeto VPL, por exemplo, tipo de projeto Web, Windows Form, Windows Service, etc. É o responsável por gerenciar a estrutura de codificação VPL que o operador implementa no componente Área de Codificação, isto, para coordenar as ações permitidas de cada componente, ou seja, o mesmo possui as regras de conexão entre os blocos da VPL e a ordem dos eventos e threads criadas. Este componente será a interface entre a VPL e o Gerador de Código. O Projeto deverá acessar o componente Configuração para verificar quais são os projetos disponíveis e componentes VPL para cada um deles, exemplo: Caso na configuração exista o tipo de projeto Windows Form, o componente Projeto deverá gerenciar e disponibilizar somente os componentes deste tipo de projeto, isto evita que componentes de um tipo de projeto seja incluído em outro e gere problemas nos componentes Geração de Código e Compilação.

- Configuração: Este componente é responsável por fornecer as configurações que correspondem ao ambiente da ferramenta VPL. É neste componente que fica as informações de quais as linguagens são suportadas para compilação, quais os tipos de projetos podem ser utilizados e os seus respectivos componentes de tela e de código suportados.

  Caso venha existir alguma extensão de algum componente e o mesmo necessite de parâmetros é nele que ficaram centralizados todas as entradas. O componente faz uso de arquivos contendo as configurações, estes estão dispostos com o recurso de include, assim a organização das configurações é transparente e independente. A estrutura dos arquivos é baseada em meta dados, definidos da seguinte forma:

  - Configurações de componentes: cada componente tem o seu espaço para parâmetros gerais, o nome para cada espaço será o próprio o nome do componente dentro do espaço de configuração de componentes. Considerar que neste espaço haverá configurações de componentes de tela, componentes de bloco de código e componentes externos;
  - Configuração de geração de código e compilação: neste espaço haverá parâmetros de configuração para cada linguagem suportada, portanto, é onde ficam registradas qual o executável e os parâmetros disponíveis para o mesmo, caso exista serviços ou executáveis auxiliares os mesmos podem ser registrados de forma recursiva, portanto as dependências podem ficar aninhadas com o executável principal.

- Área de codificação: Este componente possui a interface gráfica que representa a área de trabalho para codificação do programa que o operador deseja implementar. Esta área apresenta de forma gráfica o componente de bloco de código, que será descrito no próximo item. Além de representar os blocos de código, o componente tem a responsabilidade de notificar o componente Projeto sobre a inserção de um bloco de código, isto para que o Projeto gerencie a estrutura e as conexões de código criadas, vide descrição do componente Projeto.

- Área de protótipo de tela: Assim como na maioria das IDE este componente é a área que possibilita o operador a criar o desenho da tela do programa desejado. É nesta área que são incluídos os componentes como caixa de texto (TextBox), rótulo de texto (label), botão (button), enfim todos os componentes de tela já conhecidos em outras ferramentas de prototipação de tela de software. Para que exista a integração entre o código gerado VPL do componente Área de codificação, os componentes de tela possuem a mesma interface que os componentes de código, como pode ser visto na Figura 9, onde a classe Bloco é a classe base de todos os componentes de código e de tela. No componente Bloco de código é possível verificar melhores detalhes sobre esta estrutura.
- Bloco de código: Este componente é a base desta ferramenta VPL já que ele representa a forma como a linguagem VPL será apresentada e executada. É responsável por apresentar de maneira gráfica o código fonte do programa que o operador deseja criar. Sua estrutura possibilita que componentes de código possam ser incluídos dependendo da linguagem escolhida, ou seja, possibilita que novos componentes sejam criados e incorporados a linguagem VPL, como meio de estender as possibilidades de codificação e interatividade. Na Figura 9 pode-se notar a criação da interface ComponenteTela, será abortado no próximo item, que é uma extensão de Pacote para criação de componentes visuais que interagem com os componentes de Bloco de código, de forma que o uso seja totalmente independente, isto é, se um arquiteto, analista ou desenvolvedor alterar os componentes gráficos, ou alterar a interface ComponenteTela, isto poderá ser feito sem grandes implicações no restante do programa. Ainda na Figura 9, as classes ComboBox, TextBox, Label e Button, são classes que especializam os componentes do framework de outras linguagens, isto é, se a linguagem final desejada ser o C#, então, as classes deverão especializar as classes presentes na DLL System.Windows.Forms.dll para projetos WindowsForms ou System.Web.dll para projetos WebForms.
- Componente de tela: Este componente é a parte extensível da ferramenta de forma que sua estrutura está atrelada ao componente Bloco de código, como pode ser visto na Figura 9, a interface Componente de tela estende a

interface Pacote. Esta estrutura visa possibilitar o uso da implementação do componente Bloco de código de forma que o acoplamento entre os dois possibilite o uso de eventos de todos os componentes de tela em bloco de código. Para facilitar o entendimento, um exemplo para esta situação é o uso de um evento Click em um componente de tela dentro de um código, que executará determinadas tarefas para satisfazer os requisitos de um sistema, neste sentido esta extensão da interface Pacote faz-se necessária.

Opcionalmente sugere-se que o Componente de Tela implemente alguma interface de componente dll, jar ou lib, para reutilizar componentes já existentes no framework mais adequado para o desenvolvedor como exemplo o Microsoft Framework .Net, Spring, etc. Porém, uma das restrições deste reuso é o fato de que o uso de um framework específico limita a compilação e geração de código, pois, cada framework restringe-se ao escopo ao qual foi desenvolvido, ou seja, se for utilizado um componente desenvolvido em .Net então só será possível compilar o código desenvolvido na ferramenta neste framework.

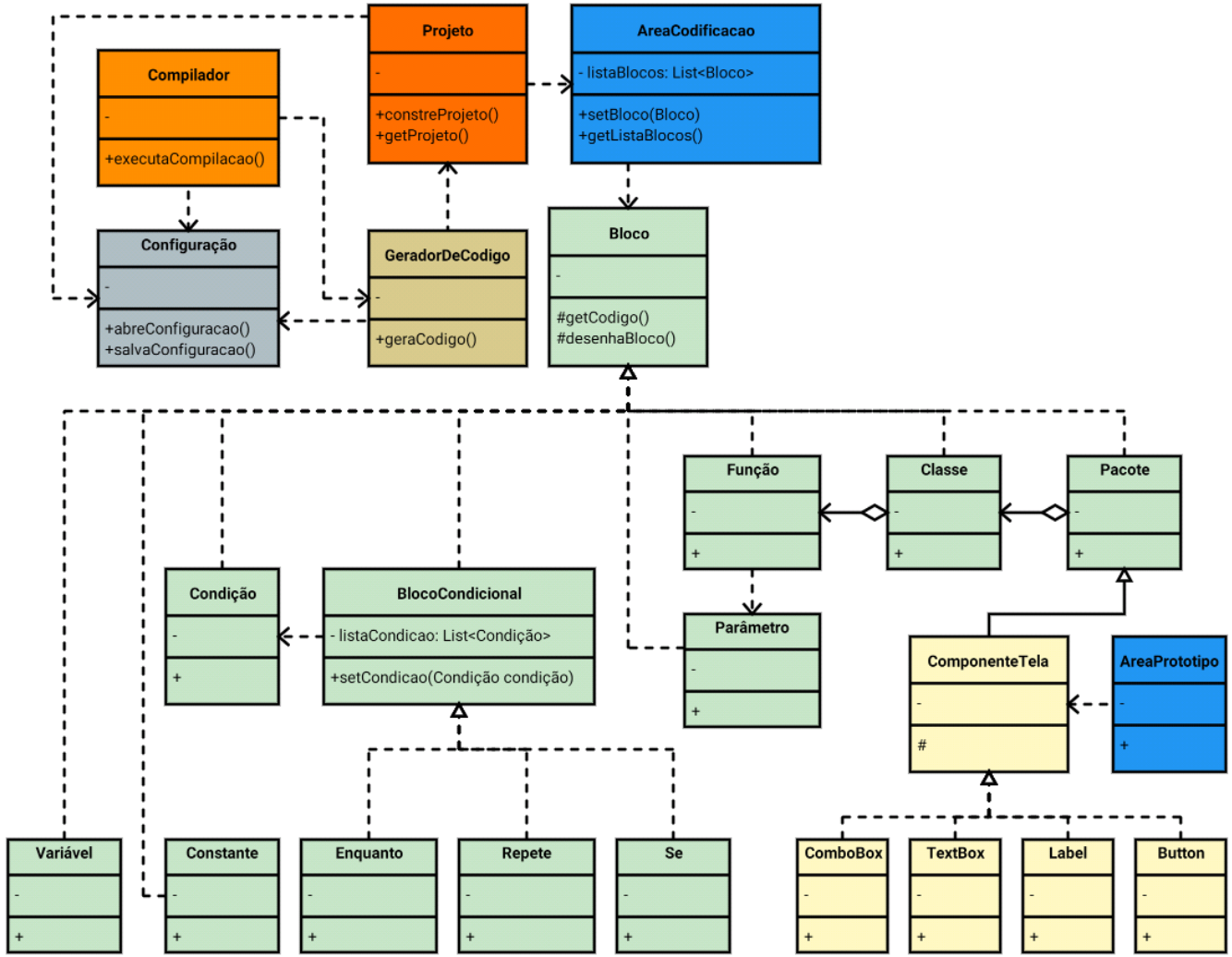
3.3 Apoio ao desenvolvimento

Como toda ferramenta de desenvolvimento ou IDE, existem recursos para facilitar o trabalho do desenvolvedor. Desta forma, a ferramenta deve possuir interface que facilite o uso de componentes visuais, desta forma o uso do recurso *drag and drop* é primordial para arrastar e soltar os Componentes de Codificação e Componentes de Tela para a Área de Codificação e Área de Protótipo de Tela respectivamente.

O uso de teclas de atalhos também são importantes para agilizar o desenvolvimento neste tipo de plataforma, já que exige grande quantidade de blocos visuais, o que acarreta em uso excessivo do mouse, e isto torna o processo de desenvolvimento moroso e cansativo, desta forma as teclas de atalho devem ser criteriosamente escolhidos para englobar a maior parte dos procedimentos repetitivos, como por exemplo a inclusão de Blocos de Código na Área de Codificação, lugares em que o desenvolvedor utilizará bastante. Desta forma combinações de teclas e caixas de diálogos inteligentes devem ser criadas, a

sugestão para o exemplo anterior de inclusão Blocos de Código na Área de Codificação é teclar [CTRL+I] e em seguida apresentar uma caixa de texto para que o desenvolvedor digite parte do nome do bloco e o sistema mostra algumas opções tal qual podem ser escolhidas com o ponteiro do mouse ou com as teclas direcionais do teclado, e ao teclar [Enter] o componente é incluído na Área de Codificação logo após o último bloco de código incluído.

Figura 9 – Modelo de domínio



4 VANTAGENS E DESVANTAGENS DA FERRAMENTA PROPOSTA

Este capítulo elucidará como esta ferramenta ajudará desenvolvedores a criar seus programas, frisando principalmente os motivos pela qual pessoas com pouco conhecimento em programação conseguirão criar seus programas para resolver os problemas do seu dia a dia. Também apresentará algumas desvantagens com relação a outras IDE tradicionais como Visual Studio e Eclipse.

4.1 Vantagens

Como essa ferramenta foi criada com conceito de VPL sua essência é a facilidade para criação de programas. Como visto no capítulo 2 algumas ferramentas VPL são criadas para uso infantil de forma que o aspecto visual tornasse o motor da ferramenta. Por conter o recurso de *drag and drop* e todos os seus componentes serem visuais e dispostos por categorias, a ferramenta torna-se mais responsiva o que proporciona melhor entendimento sobre a funcionalidade de cada objeto e mais interatividade entre o desenvolvedor os recursos disponíveis.

Na maior parte do tempo o desenvolvedor estará imerso na área de codificação, caso se sinta mais confortável e queira maior agilidade, e consequentemente mais produtividade, haverá o recurso de teclas de atalho para inserção dos blocos de código de forma rápida, sem necessidade de utilizar o *drag and drop*. Como os componentes de codificação são blocos visuais, todos os blocos inseridos estarão dentro de contexto ao qual foram submetidos, ou seja, todos os blocos inseridos dentro do contexto de um bloco *while* estarão indentados de forma que será visualmente fácil de se identificar que os comandos estão dentro deste loop, e caso dentro do bloco *while* exista a necessidade de criar um bloco *if* por exemplo, o restante dos comandos inseridos serão inseridos dentro deste *if*. Isto ajuda o desenvolvedor a se preocupar mais com a lógica de programação e menos com indentações de código, que tanto gera confusões nas linguagens comuns como C++, Java, C#, etc. Outro ponto a qual os blocos visuais dos componentes de codificação ajudam é o fato do desenvolvedor não precisar se preocupar com a sintaxe, já que, todo bloco é uma linha de código completa com a sintaxe correta,

neste sentido o desenvolvedor não se preocupa com ponto e vírgula, colchetes, chaves, etc.

Ainda na área de codificação o conceito de utilizar blocos visuais estimula o desenvolvedor a reutilizar código, pois, a ideia de bloco fica enraizado de forma que se o desenvolvedor cria um método dentro de uma classe, este método poderá ser utilizado por outros métodos em forma de blocos, isso é com certeza, nas linguagens mais comuns, uma boa prática que não é tão claro para alguns desenvolvedores, principalmente àqueles mais novos que estão no início de suas carreiras, e se considerarmos pessoas mais leigas o reaproveitamento de código é quase inexistente. Com isso, é muito provável que trechos de código sejam codificados para que possam ser reutilizados e compartilhados com outros programas.

Outro benefício desta ferramenta é a possibilidade de importação de bibliotecas externas que podem ser inseridas no código facilmente, isso possibilita que as bibliotecas existentes dentro do framework escolhido sejam utilizadas facilmente para que não seja necessário criar componentes complexos como Grid, TextBox, Label, DropDown, DataSource, etc. Considerar que os componentes que serão importados serão somente aqueles pertencentes ao framework da linguagem escolhida na criação do projeto, por exemplo, ao escolher que o código fosse compilado em C#, os componentes existentes serão apenas aqueles que pertencem ao framework do Microsoft .Net.

4.2 Desvantagens

Uma das desvantagens da ferramenta é com relação a quantidade de blocos de código visualizáveis na área de codificação, pois, quando houver métodos com muita linha de código, ficará difícil de visualizar o código já que os blocos de código ocupam uma área muito maior que um texto utilizado em ferramentas de codificação tradicionais.

Outra desvantagem é com relação a dificuldade para criar um módulo de debug para a ferramenta, pois, a ferramenta permite escolher uma linguagem para o produto final, não há como garantir que todas as linguagens tenham suporte a esta funcionalidade.

5 CONCLUSÃO

Hoje, existem muitas ferramentas que são utilizadas para prototipagem de telas que são utilizados para criar visualmente como deverá ser a interface do sistema, porém, estes protótipos possuem navegações entre telas sem regras ou atualizações de informações, ou seja, com limitações. Por outro lado, há ferramentas de desenvolvimento de software como as IDE, que possuem funcionalidades de prototipagem, mas são utilizados somente por desenvolvedores, pois, há um nível de dificuldade que desencoraja os usuários sem conhecimento técnico a tentar criar protótipos ou um software, mesmo àqueles mais simples.

A arquitetura da ferramenta proposta neste trabalho busca ajudar as pessoas com pouco conhecimento técnico em programação a criarem softwares simples. Para tanto, foi introduzido o uso do conceito de VPL, tal qual foi utilizado nos programas Scratch e Alice, e que são utilizados por crianças e adolescentes com pouco conhecimento em programação de software.

As funcionalidades das IDE foram mescladas com os conceitos da VPL, de forma que a ferramenta seja uma camada intermediária entre o código fonte baseado em texto, como C++, C# e Java, e o código VPL. Assim, a ferramenta ajudará na questão da reutilização de código, pois, todos os componentes existentes nos frameworks de cada linguagem poderão ser utilizados e estendidos, caso necessário. Nota-se que isso proporciona aos desenvolvedores e clientes a trabalharem em conjunto, assim o cliente pode iniciar o projeto com a criação dos protótipos e algumas regras e o desenvolvedor refina o código fonte para produzir o produto final.

Com isso, a arquitetura da ferramenta tem grande possibilidade de obter sucesso, caso venha ser implementado, pois, possue características e benefícios suficientes para justificar seus objetivos.

Este trabalho foi importante para conhecer conceitualmente as diferenças entre as VPE e VPL, e assim propor uma arquitetura de ferramenta que possui características tanto da VPE quanto da VPL. Além disso, pode-se perceber que a VPL não é amplamente utilizada comercialmente, mas tem potencial para que, no futuro, o desenvolvimento de software seja mais democrático, no sentido de ser

utilizado por àqueles que querem criar softwares mas tem conhecimento limitado no desenvolvimento de software.

Uma sugestão para trabalhos futuros é revisar a arquitetura proposta para incorporar ou alterar os conceitos propostos. Outra sugestão é implementar a ferramenta e utilizá-la como prova de conceito para validar ou recomendar mudanças.

6 PESQUISA BIBLIOGRÁFICA

6.1 Livros e artigos:

Majed Marji. **Learn to Program with Scratch: A Visual Introduction to Programming with Games.** Art, Science, and Math, 2014

UTTING, Ian et al. Alice, Greenfoot, and Scratch -- A Discussion. **Toce,** [s.l.], v. 10, n. 4, p.1-11, 1 nov. 2010. Association for Computing Machinery (ACM). http://dx.doi.org/10.1145/1868358.1868364.

Steve McConnell . **Rapid Development : Taming Wild Software Schedules**. Microsoft Press, 1996

MALONEY, John et al. The Scratch Programming Language and Environment. **Toce,** [s.l.], v. 10, n. 4, p.1-15, 1 nov. 2010. Association for Computing Machinery (ACM). http://dx.doi.org/10.1145/1868358.1868363.

WARREN, Scott; NORRIS, Cathleen; LIN, Lin. **Influence of Alice 3:** Reducing the Hurdles to Success in a CS1 Programming Course. 2013. 169 f. Tese (Doutorado) - Curso de Programming Course, University Of North Texas, Denton, 2013. Cap. 1. Disponível em: <http://digital.library.unt.edu/ark:/67531/metadc271795/m2/1/high_res_d/dissertation.pdf>. Acesso em: 22 maio 2016.

ZAYOUR, Iyad; HAJJDIAB, Hassan. How Much Integrated Development Environments (IDEs) Improve Productivity? **Jsw,**[s.l.], v. 8, n. 10, p.2425-2431, 1 out. 2013. International Academy Publishing (IAP). http://dx.doi.org/10.4304/jsw.8.10.2425-2431.

6.2 Web Referências

ABOUT Scratch. Disponível em: <https://scratch.mit.edu/about/>. Acesso em: 15 maio 2016.

ALICE: What is Alice?. What is Alice?. Disponível em: <http://www.alice.org/index.php?page=what_is_alice/what_is_alice>. Acesso em: 22 maio 2016.

AMBIENTE de desenvolvimento integrado. Disponível em: <https://pt.wikipedia.org/wiki/Ambiente_de_desenvolvimento_integrado>. Acesso em: 10 jul. 2016.

AVERY, James. **10 Add-Ins para tornar seu trabalho mais produtivo no Visual Studio.** [2005]. Disponível em: <https://msdn.microsoft.com/pt-br/library/cc517992.aspx>. Acesso em: 15 maio 2016.

CONSTRUCTING Kids Learning, Playing and Constructing with Children: Visual Programming Language - Infograph and Introduction. 2013. Disponível em: <http://constructingkids.com/2013/05/15/vpl/>. Acesso em: 15 maio 2016.

BARROS, Edson de Almeida Rego et al (Org.). **Alice: Uso do software No processo educacional junto aos cursos de engenharia.** 2012. Disponível em: <http://www.abenge.org.br/CobengeAnteriores/2012/artigos/104311.pdf>. Acesso em: 28 maio 2016.

CONSTRUCTIONISM (learning theory). Disponível em: <https://en.wikipedia.org/wiki/Constructionism_(learning_theory)>. Acesso em: 15 maio 2016.

CORDEIRO, Rogerio Halicki (Ed.). **Aumento de produtividade com tecnologias e ferramentas Microsoft: Proposta de valor da plataforma de.** 2013. Disponível em: <https://blogs.msdn.microsoft.com/rogerioc/2013/11/13/aumento-de-produtividade-com-tecnologias-e-ferr>. Acesso em: 15 maio 2016.

DICAS de produtividade para o Visual Studio. [2015]. Disponível em: <https://msdn.microsoft.com/pt-br/library/jj153218.aspx>. Acesso em: 15 maio 2016.

INTEGRATED development environment. Disponível em: <https://en.wikipedia.org/wiki/Integrated_development_environment>. Acesso em: 15 maio 2016.

INTELLIGENT code completion: IntelliSense. IntelliSense. Disponível em: <https://en.wikipedia.org/wiki/Intelligent_code_completion>. Acesso em: 10 jul. 2016.

LOGO (programming language). Disponível em: <https://en.wikipedia.org/wiki/Logo_(programming_language)>. Acesso em: 15 maio 2016.

RAPID Application Development. 2016. Disponível em: <https://pt.wikipedia.org/wiki/Rapid_Application_Development>. Acesso em: 15 maio 2016.

VISUAL programming language. Disponível em: <http://en.wikipedia.org/wiki/Visual_programming_language>. Acesso em: 15 maio 2016.

SCRATCH Foundation: About Us. Disponível em: <http://www.scratchfoundation.org/about-us/>. Acesso em: 15 maio 2016.

MALIK, Danijel. **VISUAL STUDIO 2015 – RELEASE DATE ANNOUNCED.** 2015. Disponível em: <http://danijelmalik.com/visual-studio-2015-release-date-announced/>. Acesso em: 22 maio 2016.

PENDSE, Vikram. **Microsoft.NET for everyone !!:** Silverlight in Visual Studio 2010. 2009. Disponível em: <http://pendsevikram.blogspot.com.br/2009_05_01_archive.html>. Acesso em: 22 maio 2016.