UNIVERSIDADE DO ESTADO DE SANTA CATARINA - UDESC

CENTRO DE EDUCAÇÃO DO PLANALTO NORTE - CEPLAN

BACHARELADO EM SISTEMAS DE INFORMAÇÃO

GILBERTO EDMUNDO TAVARES

PROTÓTIPO WEB PARA PRÁTICA DE ALGORITMOS COM ACESSO OFFLINE

São Bento do Sul, SC

2016

GILBERTO EDMUNDO TAVARES

PROTÓTIPO WEB PARA PRÁTICA DE ALGORITMOS COM ACESSO OFFLINE

Trabalho de Conclusão apresentado ao Curso de Bacharelado em Sistemas de Informação, da Universidade do Estado de Santa Catarina, como requisito para a obtenção do grau de Bacharel em Sistemas de Informação

Orientador: Prof. Dr. Luiz Cláudio Dalmolin

São Bento do Sul, SC

2016

GILBERTO EDMUNDO TAVARES

PROTÓTIPO WEB PARA PRÁTICA DE ALGORITMOS COM ACESSO OFFLINE

Trabalho de Conclusão apresentado ao Curso de Bacharelado em Sistemas de Informação, da Universidade do Estado de Santa Catarina, como requisito para a obtenção do grau de Bacharel em Sistemas de Informação

Banca Examinadora

Orientador:

Prof. Dr. Luiz Cláudio Dalmolin
Universidade do Estado de Santa Catarina - UDESC

Membros:

Prof. Dr. Nilson Ribeiro Modro
Universidade do Estado de Santa Catarina - UDESC

Prof. M.Eng. Alex Luiz de Sousa
Universidade do Estado de Santa Catarina - UDESC

São Bento do Sul SC, 28/11/2016

Dedico este aos meus familiares falecidos:
Meu irmão que sempre apoiou minha formação superior e carreira até oferecendo certa colaboração financeira.
E ainda mais recente meu pai que financiou meu pré ingresso e não pude compartilhar com ele esta conquista.

AGRADECIMENTOS

A minha mãe me apoiou em minha decisão de cursar faculdade noutra cidade. E sempre que possível (mesmo com dificuldade) ajudou como pode: com móveis para meu primeiro quarto, algumas compras de mercado e algum dinheiro quando ficava mais crítico. Inclusive isso tudo pode ter sido um dos motivos da decisão dela em vender o imóvel em que morávamos.

A toda a família Lischka que me acolheu inicialmente como hóspede na residência do casal Arnaldo e Jacira (prima da minha mãe). Foi também na empresa familiar deles, juntos com seus filhos, que tive meu primeiro emprego na cidade e me foi cedido para moradia um quartinho e dependências junto à empresa.

Ao meu irmão que me empregou as sábados e minhas cunhadas que me hospedavam, além de amigos. A todos os motoristas que me deram carona, economizando assim a passagem para que eu pudesse voltar com uma grana a mais do final de semana.

A todos os meus amigos de moradia que tiveram que me aguentar nas repúblicas que habitei, especialmente à mais recentemente extinta “Mansão Amarela”.

Ao colega acadêmico Bilecki, por ter sido para mim um exemplo de dedicação ao estudo. E que coincidentemente seguiu em sua dissertação a mesma linha que já pretendida para esta sempre que possível colaborando.

A Joseli que sua experiência passando pelo mesma situação de fim de curso foi de grande ajuda. Também seu apoio e motivação, bem como dos meu amigos de infância José e Rafael que fizeram o mesmo.

A toda a experiência social, esportiva e de conheciento que as confraternizações com outros acadêmicos, Jiudesc e os vários Latinoware proporcionaram.

As empresas Hardline e Xthor que me ofereceram estágio e emprego respectivamente. Principalmente a segunda, na qual todo aprendizado e parceria com o Richard levou ao meu crescimento profissional, acadêmico e pessoal.

Sem esquecer é claro das minhas garrafas térmicas, que mantiveram tanto café quentinho quanto suco de guaraná gelado para embalar as noites dedicadas a esta realização.

“A maioria dos bons programadores programa não por esperar ser pago ou adulado pelo público, mas porque é divertido programar.” (tradução nossa)

Linus Torvalds

RESUMO

O presente trabalho trata do desenvolvimento de um protótipo para a prática computacional de algoritmos, na aprendizagem de lógica da programação. O protótipo foi desenvolvido tendo como base a ferramenta já utilizada pela maioria do professores desta instituição que lecionam a disciplina de algoritmos. O principal diferencial apresentado neste protótipo é permitir o acesso via navegadores de internet nativamente, sem instalações adicionais. Outro ponto chave é a possibilidade de, após um primeiro acesso, utilizar esta aplicação mesmo quando sem conexão com a internet, devido a uma tecnologia que armazena os arquivos necessários no navegador. Sendo o problema a atual dificuldade de iniciar o programa para a prática durante o aprendizado de algoritmos, vem se propor esta solução para acesso simplificado independente da plataforma ou dispositivo do aluno. Foi mantida a mesma sintaxe já utilizada na ferramenta na qual foi este protótipo baseado, permanecendo a variação de Portugol criada em 2002. A interface gráfica foi recriada mantendo somente elementos essenciais, portanto minimalista. Neste ponto já se tem apresentado o protótipo visualmente, com o editor de código e passando para a simulação dos algoritmos. Conceitos de algoritmos foram estudados e principalmente seus tradutores, tanto compiladores quanto interpretadores. Foi necessário um processo de aprendizado sobre estes compiladores por não termos neste curso ainda alguma disciplina que aborde este tema. Foram utilizadas técnicas de vanguarda para funcionamento em navegadores modernos, dentre essas os últimos padrões da linguagem de programação ECMAScript (popularmente JavaScript). Nessa linguagem foram encontrados componentes de código livre: Acorn e JS-Interpreter para a análise léxico-sintática e execução semântica respectivamente, além do Ace como editor de texto.

Palavras-chaves: Lógica da programação. Algoritmos. Portugol. Interpretador.

LISTA DE ILUSTRAÇÕES

LISTA DE QUADROS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

CEPLAN	Centro de Educação Superior do Planalto Norte
CLI	*Command-Line Interface*
DOS	*Disk Operating System*
GUI	*Graphical User Interface*
HTML	*HyperText Markup Language*
IE	Internet Explorer
MS	Microsoft
PDL	*Program Design Language*
PLN	Processamento de Linguagem Natural
SW	*Service Workers*
UAL	UNESA *Algorithmic Language*
UDESC	Universidade do Estado de Santa Catarina
UNESA	Universidade Estácio de Sá

SUMÁRIO

1 INTRODUÇÃO

A base da programação é ensinar a máquina realizar determinada tarefa, para que isso seja possível é necessário que o usuário comunique-se com a máquina, numa linguagem que ela entenda, a linguagem de programação. A construção da linguagem de programação envolve ações definidas em passos que se deseja que a máquina execute, a estes passos damos o nome de algoritmos. Para isso pode-se dizer que independente do paradigma utilizado no processo de desenvolvimento ele envolve algoritmos (MEDEIROS, 2015).

Algoritmos podem não necessariamente serem computacionais, já que são um conjunto de passos finitos. Analogias simplistas são os ingredientes e modo de preparo de uma receita ou instruções para uma tarefa do cotidiano, como a troca de um pneu furado em um automóvel (MEDINA; FERTIG, 2006). Porém se tratando de computação deve ter maior formalidade, seguindo uma linguagem pois diferente do ser humano uma máquina não é tão interpretativa. Desconsideradas aqui inteligências artificiais que são desenvolvidas por algoritmos para terem comportamento de compreensão humana. Isto pode ser analogamente algo como não entender idioma estrangeiro e as instruções estarem nesta linguagem. Porém em informática isso tem que ser mais especificado com palavras ou caracteres predeterminados de início e fim, de blocos, repetições, etc (DERSHEM; JIPPING, 1990).

Sendo o berço mais popular da informática os Estados Unidos da América e seus criadores em muitos casos dessa origem ou de países com mesmo idioma as primeiras linguagens de computação e a maioria delas são em inglês (SEBESTA, 2009). Isso muda quando falamos do Portugol que possui variações, mas resumidamente sua origem vem como uma tradução de uma linguagem de programação simples e comum, com seus termos no idioma nativo do Brasil.

Tendo como foco o aprendizado de algoritmos para uma boa fundamentação de futuros programadores ou profissionais da área, muitas vezes a facilidade com o inglês não é algo comum em nosso país, a linguagem Portugol se torna um artifício vantajoso (JESUS et al., 2004). Neste sentido já é utilizado pela maioria dos professores do Centro de Educação Superior do Planalto Norte (CEPLAN) da Universidade do Estado de Santa Catarina (UDESC) livro texto e software para a prática do aprendizado de algoritmos em Portugol.

Porém quando o software é exclusivo para uso em um único sistema operacional, instalações que podem não serem simples ou mesmo não acessível de outros dispositivos, passa a complicação adicional ao aluno iniciante. Por isso o protótipo tem como proposta ser acessível em qualquer navegador web, desde que alinhado com as tecnologias mais recentes. Sem contudo deixar de permitir que possa permanecer a prática de algoritmos quando não houver conexão com a internet.

1.1 OBJETIVOS DO TRABALHO

1.1.1 Objetivo Geral

Desenvolver um protótipo de ferramenta computadorizada para prática de algoritmos, podendo realizar a criação, edição, visualização e interpretação de algoritmos em pseudo linguagem Portugol. Seu acesso deverá ser nativo a partir de navegadores web modernos, inclusive acessível quando esteja indisponível uma conexão com a internet.

1.1.2 Objetivos Específicos

Visando atingir o objetivo geral em sua totalidade lista-se a seguir objetivos específicos de modo sucinto:

- Desenvolver o protótipo;
- Fazê-lo compatível com a sintaxe atualmente utilizada;
- Permitir o acesso nativo via navegadores web modernos;
- Aplicar funcionalidade para para uso sem conexão com a internet.

1.2 JUSTIFICATIVA DO TRABALHO

No processo de ensino ao invés de abstração em papel, utilizar prática em software auxilia e muito o aproveitamento na aprendizagem. Em algoritmos é necessário o pensamento lógico, em alguns casos o aluno vêm com deficiência inclusive em lógica geral. “O Aprendizado de algoritmos não é uma tarefa muito fácil, só se consegue através de muitos exercícios” (LOPES; GARCIA, 2002, p. 1). Para esse ensino é essencial conhecer dois conceitos principais: lógica de programação e algoritmos. Primeiramente é apresentado um paralelo de situações do cotidiano com algoritmos, ao efetuar comparações de instruções passo a passo realizadas diariamente, como fritar um ovo ou trocar um pneu de automóvel.

Após o entendimento básico da lógica, passa-se então para o uso de uma ferramenta computacional para a simulação destes algoritmos, aplicando descrição na linguagem estruturada em português. Em alguns casos, utiliza-se de linguagens em inglês, como por exemplo, Pascal. Há professores que preferem usar pseudo linguagem em português somente “em papel”, iniciando a codificação diretamente em linguagem C. Tem-se inclusive, também “em papel”, de analisar e interpretar os passos do algoritmo para saber quais serão os valores em memória e a saída na tela.

Simultaneamente, com a prática escrita, sem software específico, pode-se simular os mesmos algoritmos em interpretador ou compilador. Sendo em um interpretador sua construção dividida em partes, cada uma com uma função específica. Geralmente identifica-se: o analisador

léxico, o analisador sintático, o analisador semântico, e o resultado de saída. Caso seja um compilador há, ao invés de resultado de saída, o gerador de código, que transforma o programa em linguagem de máquina composta de 0s e 1s (DELAMARO, 2004).

Diante do exposto, este trabalho justifica-se pelo empenho em facilitar o processo de aprendizagem de algoritmos, desenvolvendo um programa em navegador via Internet, fazendo com que seja desnecessária a instalação do programa, podendo ser acessado a partir de qualquer computador, com o objetivo que os exercícios sejam resolvidos com maior praticidade.

1.3 ESTRUTURA DO TRABALHO

O presente trabalho está estruturado em cinco capítulos que abordam o Processo de Ensino e Aprendizagem comum na atualidade e as Ferramentas Utilizadas, detalhes sobre o ensino com a utilização de Processamento de Linguagem Natural em Algoritmos. Após o primeiro o anterior introdutório o capítulo dois descreve principais termos tem sua teoria descrita além de apresentar as principais ferramentas utilizadas no ensino e prática de lógica de programação.

Os capítulos seguintes apresentam no três a metodologia utilizada e no quatro aplicação em seu desenvolvimento até o seu resultado, apresentando alguns temos pertinentes somente ao processo. Por fim o capítulo cinco conclui-se com as considerações finais, recomendações, sugestões e trabalhos futuros, podendo ser pelo autor ou outros interessados.

2 FUNDAMENTAÇÃO TEÓRICA

Para o entendimento e desenvolvimento de trabalho, alguns conceitos teóricos devem ser dispostos. Por exemplo o **Processamento de Linguagem Natural** (PLN) que tem menor rigidez do que se faz necessário para uma máquina, onde conjuntos de instruções podem ser descritos de modo humanamente mais natural (MEDINA; FERTIG, 2006). Exemplificando esse processamento de linguagem tem-se um conversa explicando uma rota para determinado destino ou uma receita culinária. Os demais conceitos presentes neste capítulo são algoritmos, tradutores e algumas ferramentas com proposta de prática computacional de algoritmos, no aprendizado de lógica da programação.

2.1 PROGRAMA DE COMPUTADOR

Como explica Santana (2016) programas de computador *Command-Line Interface* (CLI), tem sua interação através de um *Shell* (BASH, DOS, entre outros). Este tipo de programa é contrário à *Graphical User Interface* (GUI), que possui interação visual. A estrutura de um programa de computador de tipo CLI é $\mathrm{E} \rightarrow \mathrm{P} \rightarrow \mathrm{S}$ (entrada-programa-saída), podendo não haver argumentos de entrada. Por exemplo, um programa pode receber em sua execução, como argumento, um número inteiro para resultar em sua raiz quadrada. Outro programa já ter em seu código um número pré definido para o mesmo cálculo.

2.2 PSEUDO CÓDIGO OU LINGUAGEM

Pseudocódigos são estruturados a partir de um idioma e utilizados como *Program Design Language* (PDL), que permite descrição escrita de algoritmos com flexibilidade na estrutura. No entanto com utilização de pseudocódigo, menos alternativas são possíveis em um mesmo algoritmo. Portugol é um pseudocódigo que tem aplicabilidade no ensino de lógica da programação, desde Saliba (1992) confirma-se sua utilização para este fim. Buscando aproximar de linguagens de programação, criaram-se a partir do Portugol algumas variação de pseudolinguagem. Não há contudo um consenso na padronização, porém sensíveis são as semelhanças entre as variações. Para formalidade nos pseudo códigos ou linguagens, foi criado o inglês estruturado, que tem sua origem na linguagem de programação Pascal. Certamente outros idiomas tem suas versões, mas no caso de Portugal, Brasil e outros países de mesma origem utilizam o português estruturado denominado Portugol.

2.3 ALGORITMOS

A última das notas de Ada Lovelace, que foram republicadas em 1953, apresenta os primeiros conceitos sobre programação, descrevendo um algoritmo. Desde então estes foram

difundidos, sendo utilizados até a atualidade. Algoritmo é um termo mais amplo, sendo inclusive destinado a outras áreas e finalidades além da programação (MEDINA; FERTIG, 2006).

As definições de algoritmo encontradas na literatura são interpretativas, porém todas chegam ao mesmo resultado, um **conjunto de instruções para a resolução de uma situação**. Sendo esta a definição de algoritmo, pode então ele representar um programa de computador, apresentando operações e instruções específicas para a saída que se pretende. A Tabela 1 apresenta as estruturas representáveis em um algoritmo, de acordo com Santiago e Dazzi (2003).

Tabela 1 – Estruturas representáveis em algoritmos

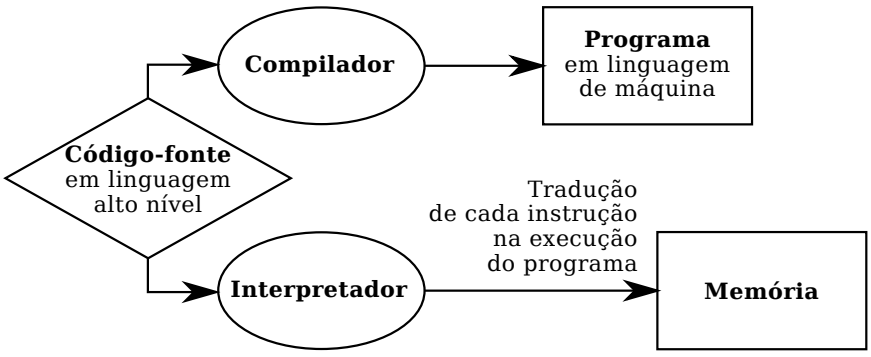
Estrutura	**Descrição**
Tipos de dados	Tipo de valores inserido nas variáveis.
Variáveis	Armazenagem na memória principal.
Atribuições	Definir valor numa variável.
Operações aritméticas	Utilizadas em cálculos entre números e variáveis.
Operações relacionais	Estabelecem relação entre comparações.
Estruturas de controle	Para fluxo, com desvio condicional e laços de repetição.
Vetores	Variáveis com único nome, contendo um índice de posições.
Matrizes	Semelhante a vetores, porém de dimensão dupla.
Funções	Subprogramas com ou sem passagem de parâmetro.

Fonte: Produção do autor, 2016.

2.4 TRADUTORES

No âmbito da computação, tradutor é um programa que traduz uma linguagem de programação para outra equivalente. Conforme Aho et al. (2007) existem dois principais tipos de tradutores: compiladores e interpretadores. A diferença entre Compilador e Interpretador pode ser compreendida na Figura 1, o interpretador na execução traduz diretamente cada instrução, partindo os dois de um mesmo código fonte.

Figura 1 – Diferença de Compilador e Interpretador



Fonte: Produção do autor, adaptado de Medina e Fertig (2006).

Passando por um processo de entendimento de elementos mínimos no código, ignorando outros e agrupando em símbolos especificados, a compilação se inicia. Além dessa análise também é necessária a verificação se esses grupos estão em ordem ou outros erros possíveis. Delamaro (2004) descreve que código fonte de entrada termina tendo como saída o programa objeto, o compilador converte a linguagem simples em uma linguagem entendida pela máquina. Na Figura 2 é possível visualizar um compilador, somente com a principal etapa e dela as análises: Léxica, Sintática e Semântica.

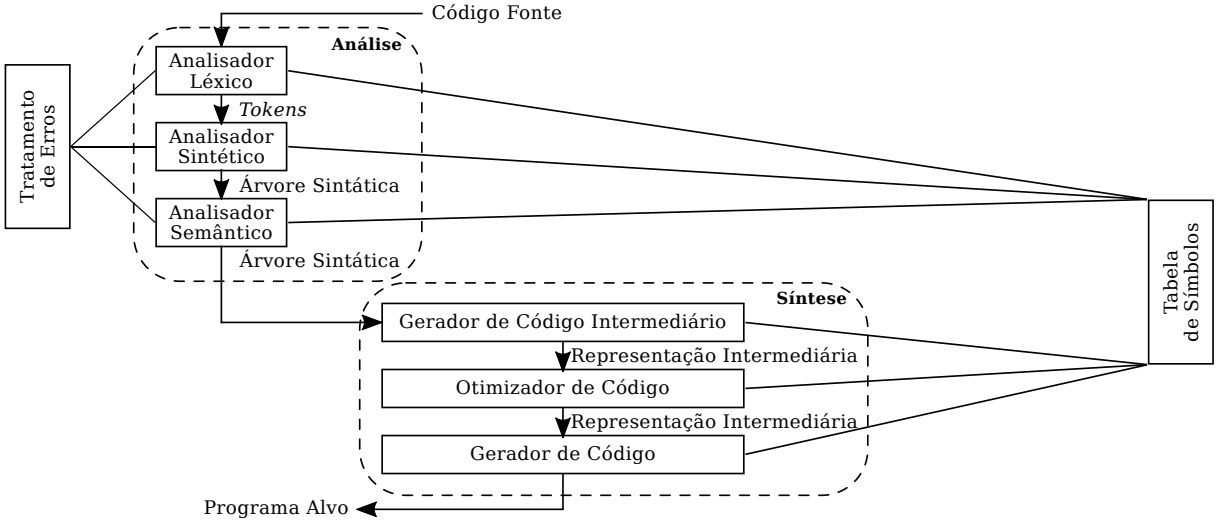
Figura 2 – Análises de um compilador

Fonte: Produção do autor, adaptado de Ferrandin e Stephani (2005).

2.4.1 Processo de compilação

Num compilador ideal, de acordo com Aho et al. (2007) e visto na Figura 3, pode-se identificar a etapa de **análise** (*front end*) e a etapa de **síntese** (*back end*). Na síntese a partir da árvore sintática são geradas representações intermediárias até o programa alvo executável através de: geração de código intermediário, otimização de código e por fim o gerador de código. Durante todo o processo a tabela de símbolos guarda informação sobre nomes declarados e o tratamento de erros que age na etapa de análise, por exemplo, identificando erro e em qual linha.

Figura 3 – Compilador completo



Fonte: Produção do autor, adaptado de Aho et al. (2007).

2.4.2 Análise Léxica

A análise léxica (*scanner*) é citada por Aho et al. (2007) como sendo onde elementos mínimos (*tokens*) são identificados, cada letra, número e outros caracteres (como pontuação e demais símbolos), além do espaço. Após reconhecidos, esses *tokens* são então agrupados em lexemas: identificadores, palavras-chave reservadas, comandos, operadores, constantes, conjunto de texto, nome de variáveis, início e fim do algoritmo ou de blocos. Essa análise também ignora comentários e marcas de edição (tabulações, caracteres de avanço de linha e espaços). No Quadro 1 é dado uma linha do algoritmo para na Tabela 2 apresenta os lexemas identificados.

Quadro 1 – Exemplo para entendimento de tradutor

```
imprima "Aprendendo Algoritmo";
```

Fonte: Produção do autor, 2016.

Tabela 2 – Lexemas encontrados no exemplo

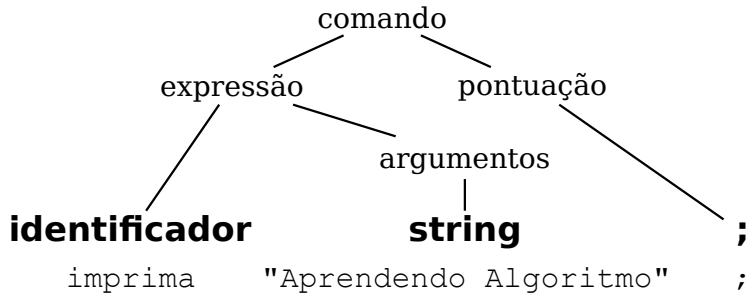
Lexema	Classificação
`imprima`	identificador da expressão de impressão
`"Aprendendo Algoritmo"`	conjunto literal de texto
`;`	caractere de encerramento de expressão

Fonte: Produção do autor, 2016.

2.4.3 Análise Sintática

Análise sintática (*parser*), recebe a sequência de lexemas da análise anterior e determina se são elementos estruturais pertencentes à linguagem desejada, exposto por Delamaro (2004). Esses elementos são estruturados de modo ramificado (Figura 4), dando origem ao termo árvore sintática. São essas estruturas: expressões aritméticas, comandos de atribuição ou declarações de procedimentos e as relações entre eles, garantindo que seja correto sintaticamente.

Figura 4 – Exemplo de árvore sintática



Fonte: Produção do autor, 2016.

2.4.4 Análise Semântica

A análise semântica verifica o "significado" das instruções, ao tratar de aspectos relacionados à sua ordem. Esta análise confere se há sentido de acordo com a gramática definida pela linguagem, para a correta execução. É possível que um programa esteja em acordo com a sintaxe gramatical, porém erros quanto a semântica são apresentados. A gramática da linguagem é utilizada nesta etapa, ela pode ser representada em uma árvore gramatical (*tree grammar*). Por exemplo se só for possível imprimir conjuntos literais de texto, variáveis, conjuntos dessas ou operações aritméticas entre parênteses e for criado como impressão direta de um número.

2.5 FERRAMENTAS PARA ALGORITMOS

Ferramentas computacionais para prática de algoritmos são utilizadas no processo de ensino aprendizagem com enfoques variados. Várias dessas ferramentas podem ser listadas:

- **ILA + AMBAP**: Desenvolvida por Evaristo e Crespo (2000) o Interpretador de Linguagem Algorítmica (ILA) é um tradutor do tipo CLI e foi utilizado para simulação dos algoritmos no editor gráfico AMBAP (AMBiente para Aprendizado e Programação);
- **UAL + EditorUAL**: Desenvolvido por Spallanzani e Medeiros (2000) na UNESA é basicamente um tradutor do tipo CLI, desenvolvido para linux. Ele foi posteriormente portado para Windows, e neste sistema operacional foi criado o EditorUAL do tipo GUI;
- **VisuAlg**: Desenvolvida por Claudio Morgado de Souza, programador e analista, bem como professor universitário, é um ferramenta utilizada tanto em cursos técnicos quanto em meio universitário. A facilidade de obtenção do seu executável para Windows e documentação completa são atrativos, além de sua simplicidade sem perder funções úteis para iniciantes (SOUZA; FRANÇA, 2013);
- **Web Portugol**: Desenvolvida por pesquisadores na UNIVALI permite o acesso via navegadores que tenham integração com Java, desde que o mesmo também esteja instalado no computador (SOUZA; FRANÇA, 2013);
- **Construtor**: Desenvolvido pelo Centro Educacional de Informática Aplicada do SENAC (Rio de Janeiro), é a primeira ferramenta para algoritmos com interface gráfica encontrada para uso em Windows. Esta ferramenta tem a vantagem de acompanhar o livro "Construção de Algoritmos" de Fernandes e Botini (1999).

Na Tabela 3 consta um comparativo de fatores que podem ser determinantes na escolha pelo educador, para que seus alunos pratiquem o conteúdo.

Tabela 3 – Comparativo das ferramentas

	UAL+Editor	ILA+AMBAP	VisuAlg	Web Portugol	Contrutor
Plataforma	Linux e Windows	DOS/Windows	Windows	Web (Java Applet)	Windows
Licença	*open source*	*freeware*	*freeware*	*open source*	*open source*
Instituição	UNESA	UFAL		UNIVALLI	SENAC
Linguagem	Haskell	Java (editor)		Java	

Fonte: Produção do autor, 2016.

Desenvolvida por Esmin (1998) quando vinculado à UNOESC, o Portugol/Plus é a mais antiga opção de ferramenta computadorizada encontrada no Brasil. Comumente é referenciada por outros autores, inclusive quando tratam sobre desenvolvimento de novas soluções. Porém por ser em plataforma *Disk Operating System* (DOS), mesmo tendo interface gráfica foi desconsiderada no levantamento da Tabela 3.

Ao delimitar a quantidade de ferramentas computacionais para prática de algoritmos, algumas foram desconsideradas, por motivo de que o acesso ao programa para testes não estava disponível: PascalX, por Arthur Vargas Lopes da ULBRA; Web-UNERJOL, utilizando UNERJOL os dois pelo mesmo acadêmico na UNERJ com colaborações distintas. Outros softwares não foram considerados por fugirem do escopo, ao utilizar robótica, jogos, foco em estrutura de dados ou similares. Nesta pesquisa eram pretendidas somente ferramentas com pseudo-linguagem algorítmica em português.

2.6 CASO DE ENSINO COM LIVRO E FERRAMENTA

O livro “Introdução à programação: 500 algoritmos resolvidos” de Lopes e Garcia (2002) tem como atrativo a grande quantidade de algoritmos resolvidos, que seu título deixa claro. Isto leva os educadores utilizarem como livro texto base de disciplinas de ensino de Lógica da Programação. Neste livro todos os algoritmos propostos para execução computadorizada estão impressos na sintaxe UAL - variante do Portugol - acompanha mídia ótica que contém os exercícios nesta e também em ILA. Na Tabela 4 é visto a comparação entre essas sintaxes.

Tabela 4 – Comparação entre UAL e ILA

Em UAL	**Em ILA**

`prog algoritmo11`
`  imprima "Aprendendo Algoritmo!!!";`
`fimprog`
	`//prog algoritmo11`
	`inicio`
	`  limpar`
	`  escrever "Aprendendo Algoritmo!!!"`
	`fim`

Fonte: Produção do autor, a partir dos exemplos de Lopes e Garcia (2002).

O presente projeto está pautado neste algoritmo simplista, obtido em Lopes e Garcia (2002, p. 26) e na mídia que acompanha o livro desses autores. Facilmente se nota a diferença

nos blocos de início e fim do programa, além de somente em UAL ter o nome do mesmo (solucionado com uso de linha comentada) e o comando de limpeza somente necessário em ILA. As chamadas de escrita em tela também têm sua palavra reservada diferente. Nenhuma delas delimita o argumento por parênteses e UAL exige ponto e vírgula ao final. As duas não são sensíveis no uso de letras maiúsculas ou minúsculas nas palavras chaves.

Tabela 5 – Escopo das linguagens UAL e ILA

	UAL	**ILA**
Saída em tela	**s**im	**s**im
Entrada de argumentos	**s**im	**s**im
Tipos de dados	**s**im	**s**im
Variáveis	**s**im	**s**im
Atribuições	**s**im	**s**im
Operações aritiméticas	**s**im	**s**im
Operações relacionais	**s**im	**s**im
Estruturas de controle	**s**im	**s**im
Vetores	**s**im	**s**im
Matrizes	**n**ão	**s**im
Funções	**n**ão	**s**im

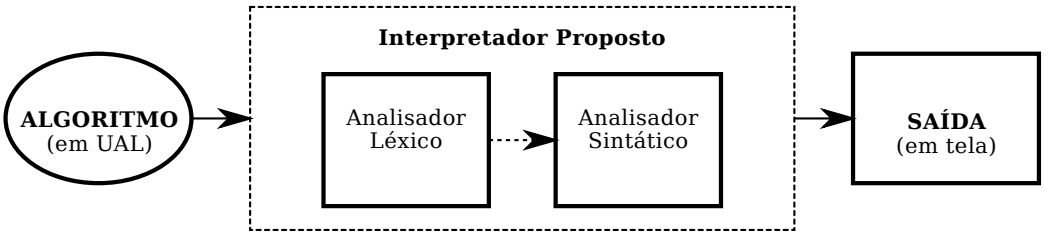
Fonte: Produção do autor, baseado em Spallanzani e Medeiros (2000), Evaristo e Crespo (2000).

Sendo as estruturas de um algoritmo o escopo da linguagem, há diferenças entre UAL e ILA também quanto sua abrangência, visto Tabela 5. Não tendo a UAL estruturas como matrizes n-dimensionais, funções com ou sem passagem de parâmetros, novos comandos para tomada de decisões e controle de repetições, novos tipos de dados, conversores de tipos (SPALLANZANI; MEDEIROS, 2000). O presente trabalho somente abrange a “Saída em tela”.

3 DESENVOLVIMENTO DA APLICAÇÃO

O desenvolvimento do protótipo neste trabalho busca ser opção para substituir o software Editor UAL, que é utilizado na UDESC/CEPLAN atualmente. Esse protótipo deve ser prático, atual, acessível e com o código disponível para que possa ser mantido e ampliado. Essencialmente a aplicação foi desenvolvida em linguagens Web, o JavaScript sendo a linguagem de programação, HTML como linguagem de marcação e CSS como estilização. A pesquisa de tradutores foi realizado, baseado em Esmin (1998), um esboço do funcionamento pretendido (Figura 5), para a simulação dos algoritmos no protótipo proposto.

Figura 5 – Tradutor proposto



Fonte: Produção do autor, 2016.

3.1 ENGENHARIA DE REQUISITOS

No desenvolvimento de software antes de iniciar a codificação, algumas etapas são esperadas e recomendadas para maior assertividade (qualidade, orçamento, prazos). Como explicado por Sommerville (2011), neste processo são acordados e especificados os detalhes que satisfazem os envolvidos. Há dois níveis de detalhamento dessa especificação, um para clientes e usuários outro aos desenvolvedores, alto nível e abrangente respectivamente. Análise de requisitos prioriza e classifica os pontos que a aplicação pretende solucionar.

Os **Requisitos Funcionais** descrevem o que o sistema deve fazer, normalmente são descritos de modo abstrato para a fácil compreensão por seus usuários (SOMMERVILLE, 2011). Puderam para o software ser levantados os requisitos funcionais vistos na Tabela 6.

Tabela 6 – Requisitos Funcionais

Código	Descrição
[RF001]	Permitir ao aluno edição de código
[RF002]	Permitir ao aluno abrir arquivos de texto
[RF003]	Permitir salvar o arquivo alterado
[RF004]	Permitir visualizar a simulação do programa

Fonte: Produção do autor, 2016.

Já os **Requisitos Não Funcionais** segundo Sommerville (2011), são relacionados não diretamente as funcionalidades, oferecidas pelo sistema aos usuários. O autor também descreve

estes requisitos sendo: confiabilidade, desempenho, proteção ou disponibilidade. Foram definidos para o software os requisitos não funcionais presentes na Tabela 7.

Tabela 7 – Requisitos Não Funcionais

Código	Descrição
[RNF001]	O programa deve prezar pelo menor tamanho possível nos arquivos
[RNF002]	O sistema deverá rodar em navegador (*browser*) moderno
[RNF003]	Deverá ter interface minimalista, mas visualmente agradável
[RNF004]	Ser acessível mesmo que desconectado da internet

Fonte: Produção do autor, 2016.

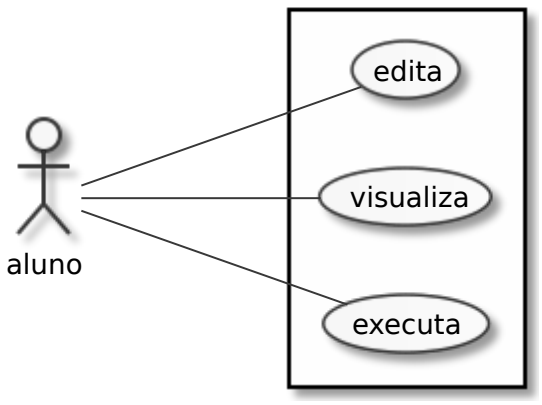
3.2 DIAGRAMAS UML

Na etapa de **Projeto Lógico** tem-se a elaboração de documentos do projeto, fase importante para melhor entendimento do sistema em desenvolvimento. Inclui-se nas seguintes subseções os diagramas UML, foi desenvolvido o diagrama de casos de uso.

3.2.1 Diagrama de Casos de Uso

De acordo com Guedes (2011) este é o diagrama mais geral da UML, seu modo informal faz ser um dos primeiros a ser utilizado no levantamento e definição de requisitos. Costuma ser consultado também durante o processo, inclusive base para outros diagramas. Colabora ainda para que os usuários tenham ideia geral do funcionamento proposto, por sua simplicidade e fácil compreensão. Nesse diagrama são identificados atores (pessoas, sistemas, hardware) e funcionalidades, nele denominadas casos de uso. De modo simplificado e com clareza a Figura 6 apresenta as atividades do software.

Figura 6 – Caso de uso Aluno



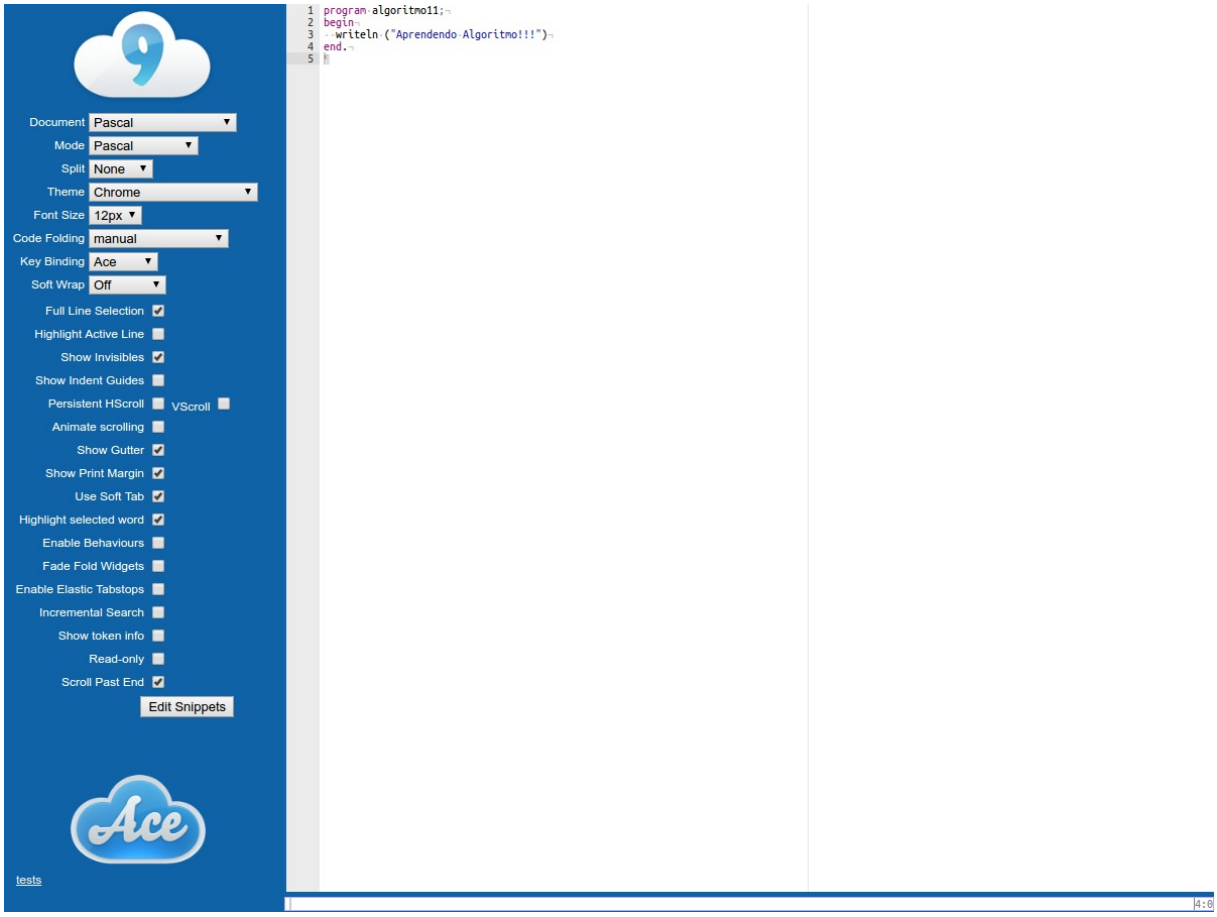
Fonte: Produção do autor, 2016.

3.3 EDITOR DE CÓDIGO ACE

No modelo em que é possível embutir nas páginas web ou aplicações via navegador, já existem editores de código fonte de linguagens de programação. Dentre esses editores de código em texto, os mais utilizados tem-se o ACE e o CodeMirror. Foi optado pelo ACE, pela praticidade de sua adição no protótipo e fácil criação do modelo gramatical para destaque de palavras reservadas da linguagem.

Em tecnologia há áreas para experimentações com determinada ferramenta ou tecnologia (*playground*). No caso do editor ACE é disponibilizado uma "pia de cozinha" (*Kitchen Sink*) conforme Figura 7, onde o editor embutido pode ser modificado através de um menu lateral, quanto a linguagem ou tema e outras de suas várias opções.

Figura 7 – Ace *Kitchen Sink*



Fonte: Captura de tela da parte visível no navegador.

Inicialmente as funcionalidades pretendidas são todas supridas e outra mais levam a esta ser uma opção útil para evolução da aplicação em trabalhos futuros. Ignorando as duas que dizem respeito ao destacamento de sintaxe em mais de 110 linguagens (customizável, o que permitiu adicionar UAL) e mais de 20 temas (também personalizado para o mesmo estilo encontrado no Editor UAL), as demais funcionalidade estão expostas no Quadro 2.

Quadro 2 – Demais Funcionalidade do Editor ACE

Descrição
Identação automática
Permite documentos com muitas linhas
Atalhos de teclado personalizáveis
Localizar e substituir, podendo utilizar expressões regulares
Alternar entre tabulação suave (espaços) ou tabulação real
Exibir caracteres ocultos
Arrastar e soltar utilizando mouse
Limitação de colunas no texto, com quebra
Retrair ou expantir código
Cursores e seleção múltiplas
Checagem em tempo real para algumas linguagens
Funcionalidade de recortar, copiar e colar

Fonte: Produção do autor, 2016.

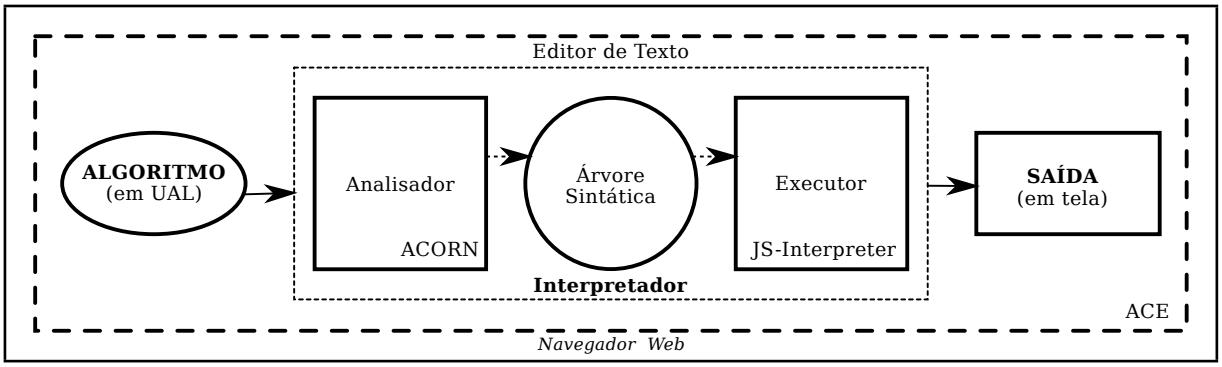
3.4 MÓDULOS PARA TRADUÇÃO

Para resultar na árvore sintática encontrou-se o **ACORN**, um parser já escrito em JavaScript. De modo trabalhoso foi necessário adaptar o código fonte para compreender a estrutura do algoritmo proposto. Principalmente devido à não utilização de parênteses para o comando de impressão em tela, em Javascript `document.write("Olá!");` e UAL `imprima "Olá!";`

Para a saída simulando o algoritmo o **JS-Interpreter** foi encontrado como opção. Ele já se utiliza do mesmo ACORN mencionado acima e interpreta a própria linguagem Javascript. Pelo mesmo motivo citado no parágrafo anterior, devido às diferenças de sintaxe entre Javascript e UAL, se tornou uma tarefa um tanto dispendiosa fazer com que ele fosse adaptado ao protótipo.

Em conjunto o Acorn e o JS-Interpreter podem ser vistos na área central da Figura 8, funcionando como tradutor para sintaxe UAL resultando na saída em tela simulada, processo indicado como Interpretador. Também é possível verificar na Figura 8 outros elementos do software: navegador web, editor de texto.

Figura 8 – Esquemático do Interpretador

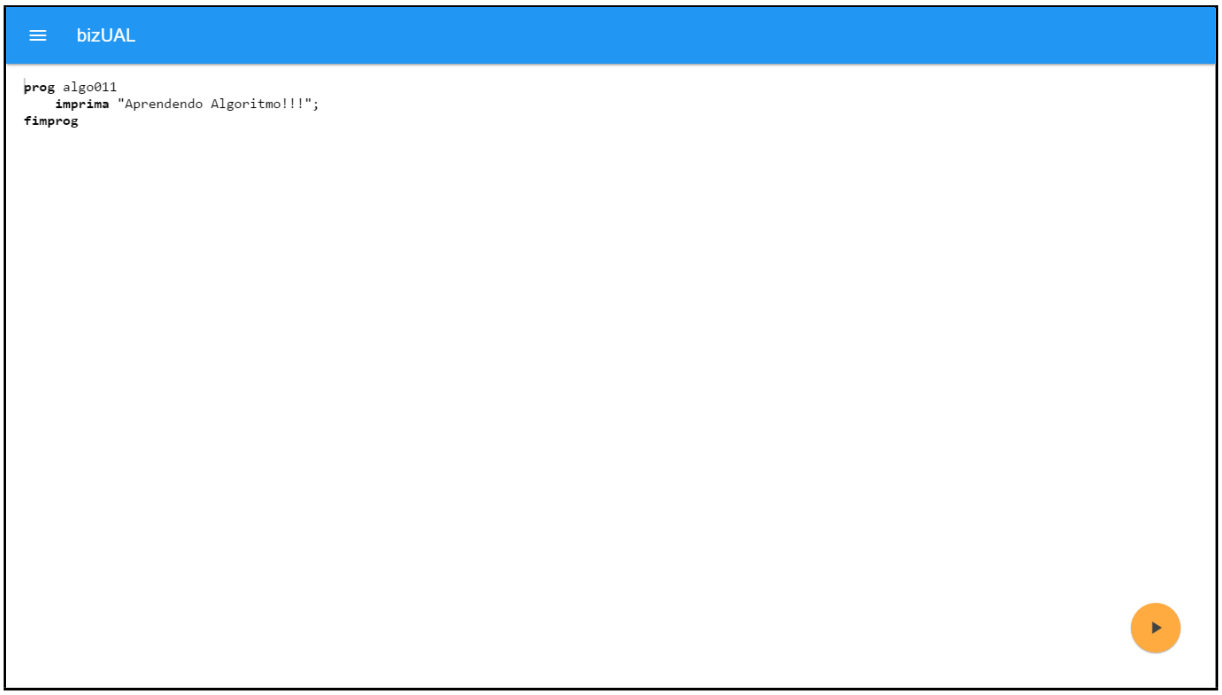


Fonte: Produção do autor, 2016.

3.5 INTERFACE GRÁFICA

Como interface gráfica foi mantido o mínimo necessário. Somente o editor de texto e um botão para executar. Foram visualizadas renderizações e tamanhos de telas variados, computadores (Figura 9), tablets, smartphones, etc. Seguindo conceitos do Google Material Design foi adicionada uma barra no topo como botão para menu o nome da aplicação, denominada Bizu, e um botão de ação flutuante no canto inferior direito como ícone “play” para a execução do algoritmo.

Figura 9 – Interface Desktop



Fonte: Produção do autor.

Também na Figura 9 pode ser notado o nome “bizUAL”, este foi utilizado para facilitar a identificação do software entre alunos e professores. A escolha do nome surgiu a partir da palavra bizu adicionada a sigla da sintaxe UAL. A gíria “bizu” teve origem em quartéis militares, onde os experientes sussurravam dicas para os novatos, os superiores de longe ouviam aquela onomatopéia característica de cochicho “bzbzbzu”, dando origem ao termo.

Facilitando a tarefa de construção da interface foi utilizada a biblioteca Material Design Lite (MDL) disponibilizada oficialmente pelo Google. Essa biblioteca possui em seu site uma funcionalidade, na qual cores foram escolhidas.

3.6 ACESSO *OFFLINE*

Com auxílio de ***Service Workers*** (SW) é possível resolver a navegação *offline*. Está sendo desenvolvido, como explica Archibald (2014), por um esforço colaborativo entre grandes empresas como Google, Mozilla e outras, sua implementação tem sido constante principalmente nos navegadores Chrome e Firefox. Realmente estimulante para os interessados na competição entre *web* e aplicações nativas nos sistemas operacionais.

Sua implementação no modo mais básico em uma aplicação de página única pode ser considerada simples. Porém sua versatilidade e aplicabilidade é incrivelmente mais extensa. Inicialmente é necessário informar ao navegador o arquivo Javascript que será o controlador, este arquivo se comporta como um *proxy* local. Para indicar o arquivo ao navegador utiliza-se o comando de registro (presente no Quadro 3), após testar se há compatibilidade.

Quadro 3 – HTML destacando o registro do SW

```
<!DOCTYPE html>
<html>
  <head>
    (...)
  <body>
    (...)a
    <script>
      if (navigator.serviceWorker !== undefined) {
        navigator.serviceWorker.register('sw.js');
      }
    </script>
</html>
```

Fonte: Produção do autor, 2016.

Dentro do arquivo `sw.js`, notável em Quadro 4, constam dois eventos: um de instalação para quando o navegador realiza a instalação (***install***) de acordo com o registro e outra para captura de ação de rede (***fetch***), onde iria naturalmente requisitar determinado recurso. O evento de instalação lista os arquivos a serem armazenados e o de rede manipula a utilização os arquivos armazenados.

Durante o trabalho tem sido utilizada a denominação "navegador moderno", para nosso estudo são navegadores que apostam em novas tecnologia que estejam surgindo. Normalmente podem ser considerados o Google Chrome, Mozilla Firefox e Opera. A Microsoft vêm flexibilizando essas adoções nas últimas versões e do seu substituto o Edge, porém quando tecnologias ainda em rascunho ela opta por sua definição e maturidade. A Tabela 8 apresenta como está, mesmo que parcial, essa compatibilidade. Ressaltando o quão recente é essa adoção, as versões lançadas somente neste ano como estáveis entre 9 de setembro e 23 de outubro, são nesses navegadores a primeira implementação ocorrida. Apenas o Google Chrome para desktop possui compatibilidade nas versões anteriores, desde 2 de maio também de 2016.

Quadro 4 – Conteúdo do arquivo SW

```
self.oninstall = function (event) {
  event.waitUntil(
    caches.open('bizual-static-v1').then(function (cache) {
      return cache.addAll([
        './',
        'css/all.css',
        'js/material.min.js',
        'js/aceual.js',
        'imgs/icon.png'
      ]);
    })
  );
};

self.onfetch = function (event) {
  event.respondWith(
    caches.match(event.request)
  );
};
```

Fonte: Produção do autor.

Tabela 8 – Suporte à SW pelos navegadores

Edge	Firefox	Chrome	Safari	Opera	iOS	Android
	49+	49+		41+		5.0+

Fonte: Produção do autor, baseado em Caniuse.com (2016).

Algumas observações são necessárias sobre a Tabela 8, o Microsoft **Edge** está nessa compatibilidade como "em desenvolvimento", e o Internet Explorer (IE) foi intencionalmente ignorado por certamente nunca receber essa implementação já que seu desenvolvimento foi descontinuado em favor do Edge. O **Safari** da Apple utiliza motor Webkit e ele está com a situação, quanto essa compatibilidade, como "sob consideração". E o **Android** 4.4.4 já utilizava WebView do Chrome mas só na 5.0 veio nativa e independente, porém instalar a última versão do Google Chrome para esse sistema operacional *mobile* tem o mesmo efeito, desde WebView ou o navegador Google Chrome para Android na versão 53.

4 CONSIDERAÇÕES FINAIS

O objetivo principal deste projeto era apresentar o conceito de aplicação para prática de algoritmos sem instalação, mas permitindo uso *offline*. Neste sentido sua estrutura como tradutor se mostrou desafiadora mesmo após a opção por componentes para montagem de editor e analisadores. Por outro lado, mesmo recente a tecnologia de tratamento atual para acesso desconectado é bem acessível quanto a explicação de sua utilização, considerado como de dificuldade moderada.

Notou-se que há ruptura devido às possibilidades de ferramentas e variações nas suas sintaxes de pseudo-linguagens. Portugol podia ser objetivo quanto à sua estrutura e não há ainda algum trabalho em prol de sua especificação.

Tentativas anteriores foram realizadas para disponibilização de acesso via navegador por ferramenta similar, com uso de *Applet* Java. Porém o iniciante não costuma ter instalado em sua máquina a *Java Virtual Machine* (JVM) e mesmo os navegadores como o Google Chrome tem perdido o suporte a essa tecnologia.

Devido a análise do cenário pode-se ter além do acesso *offline*, a primeira iniciativa de âmbito educacional em ensino superior, considerando este importante nicho dos dispositivos móveis. É gratificante ter como resultado uma aplicação que pode ser utilizada em qualquer dispositivo móvel, já que é raro quem não tenha um smartphone (principalmente entre universitários), mesmo sem se utilizar das linguagens nativas de seus sistemas operacionais.

4.1 RELAÇÃO ENTRE OS OBJETIVOS E OS RESULTADOS

O protótipo foi desenvolvido em acordo com o que consta consta definido nos objetivos, sua sintaxe é compatível com UAL, contendo a primeira estrutura “saída em tela”. Esse desenvolvimento foi realizado de maneira pública no Github e está disponível para livre utilização, constando nas referências deste trabalho.

Permite o protótipo acesso via navegadores web modernos, mesmo sem conexão com internet e não é necessária qualquer instalação adicional. Esperava-se que para tais fins que fosse mais ampla a compatibilidade, contudo os navegadores compatíveis são os mais populares. Principalmente a Microsoft deveria em breve implementar essa funcionalidade para *Service Worker* em seu navegador (Edge, substituindo o IE) pois é considerado um software promissor que estaria atualizado com as tecnologias, além de que vem pré instalado como navegador padrão em seu popular sistema operacional Windows (na versão 10).

4.2 LIMITAÇÕES DA PESQUISA

Como já citado principalmente quanto a tecnologias não foi produtivo a procura em bases científicas. Mas no limite nacional a Sociedade Brasileira de Computação (SBC) mostrou-se grande apoiadora devido aos vários artigos relacionados aos congressos, *workshops* e outros promovidos por essa instituição que puderam ser obtidos.

Dentre o maior tema e também por relação com este, informática na educação que proporcionou material importante para composição dos capítulos presentes nesta dissertação. Quanto pesquisa por conteúdo, no que se refere em encontrar trabalhos que contivessem construção de software, teve resultados mínimos. Estes materiais não possuíam em sua estrutura boa descrição de seu desenvolvimento, podendo ter essa situação influenciado no resultado do presente trabalho.

Como entusiasta e evangelizadors do compartilhamento de informações e conhecimento em software livre, é triste perceber que inclusive em grandes universidades públicas não há aderência a esta filosofia, pois de alguns softwares os códigos não foram encontrados. Lembra-se para que não seja lido aqui como gratuito, pois pode e é por muitas empresas utilizado esse pensamento de maneira comercial gerando bons lucros.

4.3 LIMITAÇÕES DO TRABALHO

Por seguir um processo diferenciado de construção este protótipo limita-se no uso das estruturas de algoritmo, não compreendendo estruturas como vetores, matrizes, funções, abordando exclusivamente algoritmos sem estrutura. São possíveis no protótipo apenas algoritmos simples de saída em tela, pois o foco era a viabilidade e disponibilidade independente de sistema operacional ou instalações, mesmo que *offline*.

Outra limitação foi a não possibilidade de gravar e manter executáveis da saída gerada do algoritmo, entendeu-se que este ponto não essencial para a prática do conhecimento e que uma possível solução no modelo pretendido seria dispendioso e demorado.

Não há a armazenamento dos algoritmos em servidor, por depender de estrutura adicional como hospedagem, além de programação adicional o devido tratamento desta funcionalidade. Ficando por conta do aluno o salvamento, podendo recorrer a algum serviço online ou mesmo memórias flash (pen drive).

4.4 SUGESTÕES PARA TRABALHOS FUTUROS

Oferecendo inúmeras possibilidades é desejado que o presente trabalho seja uma inspiração para novos projetos, ligados diretamente ou mesmo só pela tecnologia para uso *offline* no

presente descrita. É possível listar algumas, tentando enumerá-las em uma sequência lógica:

1. Ampliar o escopo das estruturas de algoritmos;
2. Acesso via login e senha com repositório dos códigos;
3. Desenvolver linguagem que una e venha a substituir a UAL buscando similaridade com Java, mas mantendo a retrocompatibilidade opcional;
4. Permitir flexibilidade para definir as variações de Portugol para que possa ser utilizado por usuários de outros ambientes.

REFERÊNCIAS

ACE. **Ace Kitchen Sink**. 2016. Acesso em: 24 agosto 2016. Disponível em: <https://ace.c9.io/build/kitchen-sink.html>.

AHO, Alfred V. et al. **Compiladotes, princípios, técnicas e ferramentas**. 2. ed. São Paulo: Person, 2007. ISBN 8588639246.

AJAX.ORG. **Ace - The High Performance Code Editor for the Web**. 2016. <https://ace.c9.io/>. Acesso em: 24 agosto 2016.

ARCHIBALD, Jake. **Service Worker - first draft published**. 2014. <https://jakearchibald.com/2014/service-worker-first-draft/>. Acesso em: 4 novembro 2016.

CANIUSE.COM. **Can I use... Service Workers**. 2016. <http://caniuse.com/#feat=serviceworkers>. Acesso em: 4 novembro 2016.

DELAMARO, Márcio Eduardo. **Como Construir um Compilador Utilizando Ferramentas Java**. São Paulo: Novatec, 2004. ISBN 8575220551.

DERSHEM, Herbert L; JIPPING, Michael J. **Programming languages: structures and models**. [S.l.]: Wadsworth Publ. Co., 1990.

DICIONARIOINFORMAL.COM.BR. **Significado de Bizu**. 2016. Acesso em: 27 setembro 2016. Disponível em: <http://www.dicionarioinformal.com.br/bizu/>.

ESMIN, Ahmed Ali Abdalla. Portugol/plus: uma ferramenta de apoio ao ensino de lógica de programação baseado no portugol. In: **IV Congresso RIBIE**. Brasília: Anais do IV Congresso RIBIE, 1998.

EVARISTO, Jaime; CRESPO, Sérgio. **Aprendendo a Programar - Programando numa Linguagem Algorítmica Executável (ILA)**. [S.l.]: Book Express, 2000. ISBN 8586846473.

FERNANDES, Antonio Luiz Bogado; BOTINI, Joana. **Construção de algoritmos**. Rio de Janeiro: SENAC, 1999. ISBN 8585746564.

FERRANDIN, Mauri; STEPHANI, Simone Lilian. Ferramenta para o ensino de programação via internet. In: **Congresso Sul Catarinense de Computação**. Criciúma: Anais do I Congresso Brasileiro de Computação, 2005. p. 102–110.

FRASER, Neil. **JS-Interpreter - A sandboxed JavaScript interpreter in JavaScript**. 2016. <https://github.com/NeilFraser/JS-Interpreter>. Acesso em: 29 agosto 2016.

FUEGI, J.; FRANCIS, J. Lovelace babbage and the creation of the 1843 'notes'. **IEEE Annals of the History of Computing**, v. 25, n. 4, p. 16–26, Oct 2003. ISSN 1058-6180.

GOOGLE. **Material Design Lite**. 2016. <https://getmdl.io/>. Acesso em: 19 julho 2016.

GUEDES, Gilleanes T. A. **UML 2 : uma abordagem prática**. 2. ed. São Paulo: Novatec, 2011. ISBN 978-85-7522-281-2.

HAVERBEKE, Marijn. **Acorn - A small, fast, JavaScript-based JavaScript parser**. 2016. <https://github.com/ternjs/acorn>. Acesso em: 29 agosto 2016.

HAVERBEKE, Marijn. **CodeMirror - In-browser code editor**. 2016. <https://codemirror.net/>. Acesso em: 24 agosto 2016.

JESUS, E. A. de; SANTIAGO; DAZZI, R. L. S. R. de. Ferramenta para criação e teste de algoritmos utilizando fluxogramas ou portugol. In: **Congresso Brasileiro de Computação, 2004, Itajaí**. Itajaí: Congresso Brasileiro de Computação, 2004.

LOPES, Anita; GARCIA, Guto. **Introdução à programação: 500 algoritmos resolvidos**. Rio de Janeiro: Campus, 2002. ISBN 9788535210194.

MEDEIROS, Antônio Vinícius Menezes. Um interpretador online para a linguagem portugol. 2015.

MEDINA, Marco; FERTIG, Cristina. **Algoritmos e programação**. São Paulo: Novatec, 2006.

SALIBA, Walter Luiz Caram. **Técnicas de Programação: Uma Abordagem Estruturada**. São Paulo: Markon, 1992. ISBN 0074607316.

SANTANA, Igor. **Criando aplicações CLI utilizando Node.js**. 2016. <http://tableless.com.br/criando-aplicacoes-cli-utilizando-node-js/>. Acesso em: 29/11/2016.

SANTIAGO, Rafael de; DAZZI, Rudimar Luís Scaranto. Ferramentas que auxiliam o desenvolvimento da lógica de programação. In: **XII SEMINCO - Seminário de Computação, 2003, Blumenau. Anais do XII SEMINCO**. Blumenau: Furb, 2003. p. 113–120.

SEBESTA, Robert W. **Conceitos de linguagens de programação**. Porto Alegre: Bookman, 2009. ISBN 8577808629.

SOMMERVILLE, lan. **Engenharia de Software**. 9. ed. São Paulo: Pearson, 2011. ISBN 978-85-7936-108-1.

SOUZA, Márcia Valéria Rocha de; FRANÇA, A. César C. Ferramentas de auxílio ao aprendizado de programação: Um estudo comparativo. In: **XIII ERBASE - WEIBASE**. Aracaju: Anais da XIII ERBASE - WEIBASE, 2013. p. 41–50.

SPALLANZANI, Adriana Sayuri; MEDEIROS, Andréa Teixeira de. **UAL Unesa Algoritmic Language**. Rio de Janeiro: Unesa, 2000.