CENTRO DE EDUCAÇÃO PROFISSIONAL HERMANN HERING

CURSO TÉCNICO EM INFORMÁTICA

Matrix
Autocomplete

Gustavo Henrique Spiess

Lucas Gabriel da Costa

Blumenau (SC)

2016

CENTRO DE EDUCAÇÃO PROFISSIONAL HERMANN HERING
CURSO DE TÉCNICO EM INFORMÁTICA

Gustavo Henrique Spiess
Lucas Gabriel da Costa

Matrix
Autocomplete

Trabalho de Conclusão de Curso apresentado como requisito para conclusão de curso.

Orientador: FABIO BUSNARDO

Blumenau
2016

DEDICATÓRIA

(ESTA PÁGINA NÃO EXIBE TÍTULO)

AGRADECIMENTOS

Ao(à) _____,
pelos conselhos sempre úteis e precisos com que, sabiamente, conduziu este trabalho.

EPÍGRAFE

(ESTA PÁGINA NÃO EXIBE TÍTULO)

“Simsimsimsimsimsimsimsimsimsi
msimsimsimsimsimsimsimsimsimsimsimsi
msimsimnsimsimsimsimsimsimsimsim.
”

- indicação de autoria -

RESUMO

Esse trabalho é a exploração da possibilidade do uso dos conceitos de software evolutivo na construção de um autocomplete, com a descrição da construção de uma biblioteca genérica para tal metodologia, de forma que pode ser reutilizada po'steriormente. O foco não é propriamente a construção de uma aplicação, mas a descrição de uma metodologia, suas características positivas, negativas e seus limites. Os esforços, apesar de não terem conseguido atingir as soluções mais otimizadas tiveram sucesso em demonstrar a execução de dois tipos diferentes de análises na aplicação do jovem campo dos algoritmos evolutivos, como solução de problemas e método de busca.

Palavras-chave e/ou descritores: Computação evolutiva; Software evolutivo; Algoritmo evolutivo; Autocomplete; Caixeiro-viajante;

SUMÁRIO

Sumário

1 INTRODUÇÃO

Os conceitos de Inteligência artificial, software evolutivo, e autocomplete não são novos, e, muito provável, já se cruzaram antes, possivelmente com resultados mais efetivos do que os dispostos nesse trabalho.

Mas o desenvolvimento desse trabalho não tem objetivos práticos, apenas interessando-se na experimentação e pesquisa de uma alternativa, construindo um código que possa ser reutilizado. Pois o foco não está voltado para o autocomplete em si, mas sim para a metodologia utilizada na sua construção.

1.1 JUSTIFICATIVA

Existem hoje no mercado inúmeras soluções para autocorrect e autocomplete. Estes dois termos, em suma, significam quando seu celular, tablet ou computador tenta prever o que você está escrevendo. A ideia em si é simples, perceber um padrão de escrita do usuário e correspondendo a ele sugerir correções e completar as palavras digitadas.

As soluções mais modernas para este quesito usam de inteligência artificial avançada para compartilhar os vícios linguísticos de diversos usuários, até mesmo sugerindo palavras que este ou aquele usuário nunca usaram. Isso acontece por que outros usuários já a usaram em contextos equivalentes. Por outro lado, não raro, as sugestões fazem com que as frases percam o sentido.

A proposta do nosso grupo é um sugestor para completar, mas não corrigir as palavras. Ele teria algumas características distintas, como por exemplo ele não fará uso de dicionários pré-estabelecidos, fazendo a construção de um dicionário interno conforme uso. Alguns dos problemas que isso cria são erros ortográficos provenientes do usuário sendo replicados conforme o uso, e em um primeiro momento não haveriam sugestões funcionais.

Os sugestores mais simples podem ser encontrados no Excel e no notepad++. O funcionamento destes usa como base as palavras digitadas no

mesmo arquivo que se está digitando. Ao usar muitas vezes o termo “procrastinação” em um arquivo qualquer sendo editado em uma destas plataformas, ao começar a digitar “propriedade” a primeira sugestão seria “procrastinação”.

Os sugestores mais complexos como os usados em celulares fazem algo semelhante a isso, mas com uma abrangência maior (trabalhando com frases e expressões) e uma maior largura na fonte dos dados (usando de todas as frases digitadas).

1.2 OBJETIVOS

A intenção do grupo é fazer a construção de um protótipo de um aplicativo que sugira possíveis continuações para uma frase. E evoluam as sugestões com uma seleção artificial e uma reprodução espelhada à biológica.

1.2.1 GERAL

Construir um protótipo de inteligência artificial que evolua na sugestão de palavras para continuidade de frases. Com um código pseudoaleatório para imitar a evolução biológica.

1.2.2 ESPECÍFICOS

- Inserir novas palavras conforme uso.
- Mesclar eficientemente os códigos de dois ou mais objetos, adicionando um fator aleatório.
- Simular o uso sem necessidade de uma pessoa.
- Manter em uso os objetos cujos genomas sejam mais eficientes, tirar os menos eficientes, e repopular com mesclas dos mais eficientes.

2 FUNDAMENTAÇÃO

2.1 O que é Inteligência Artificial

A inteligência artificial segundo Gongora (2007), é um conceito de comportamento inteligente em construtos humanos, buscando elevar isso à réplica de um comportamento humano.

“Pretende-se, com ela, capacitar o computador de um comportamento inteligente. Por comportamento inteligente devemos entender atividades que somente um ser humano seria capaz de efetuar. Dentro destas atividades podem ser citadas aquelas que envolvem tarefas de raciocínio (planejamento e estratégia) e percepção (reconhecimento de imagens, sons, etc.), entre outras. ”

Isso é, um software que apresenta comportamentos tipicamente humanos, como identificar um rosto em uma foto, são modelos de inteligência artificial.

Também segundo Gongora (2008) IA “É um conceito relativo à construção de estruturas cognitivas do ser humano, responsáveis pela formação da razão, característica peculiar frente aos demais animais. Como o ser humano é o único animal racional, diz-se que ele é o único ser inteligente. Há estudos que atribuem o conceito de inteligência a outros animais e vegetais. Mas, obviamente, não é um conceito comparável ao da inteligência humana. É, isso sim, um conceito relativo à análise em questão: esta inteligência irracional seria a capacidade de adaptação de um ser vivo às circunstâncias de seu meio. Desta forma, poderemos utilizar este conceito para a máquina, definindo, então, uma inteligência de máquina. ”

Portanto, o conceito de inteligência de máquina é também aplicável aos objetivos propostos para esse trabalho, como no tratamento de novas palavras, adicionando-as aos dicionários.

Outros conceitos descritos são as tarefas especializadas, que “possui objetivos e aplicações bem específicos, dentro de um universo delimitado. ”

“Quando queremos realizar aplicações mais práticas, estamos trabalhando com tarefas especialistas, que são aquelas aplicadas em alguma

profissão, resultado da síntese do conhecimento de especialistas no assunto. Daí surge o termo sistemas especialistas".

E as tarefas formais: "Há um conjunto de tarefas que exige análises de possibilidades para chegar-se a uma solução. Este tipo de tarefas são chamadas de formais por terem uma aplicabilidade apenas em áreas fora do mundo real, tais como jogos e problemas matemáticos. No caso de jogos, torna-se necessária a organização das ações e reações dos jogadores, todas elas previsíveis e, portanto, manipuláveis dentro de regras claras. Nos problemas matemáticos, é semelhante, havendo regras para a construção de formas geométricas, encadeamento de expressões de lógica e derivação no cálculo integral, entre outras aplicações. Estas são, portanto, aplicações situadas dentro de contextos abstratos com variáveis previsíveis. "

Aplica-se ao problema proposto a segunda descrição, isso é as tarefas formais. É um problema com regras delimitadas, onde a interação do usuário se aplica dentro de um universo pequeno, onde tudo o que ele poderá fazer é a introdução de palavras e a seleção de sugestões.

2.2 O que é algoritmo evolutivo:

No artigo *Computação Evolutiva: Uma Abordagem Pragmática* São descritos os conceitos da aplicação de um algoritmo (ou software) evolutivo na prática.

"O software evolutivo é aplicar o processo de evolução natural como um paradigma de solução de problemas, a partir de sua implementação em computador" Zuben (2000).

Ela baseia-se no conceito de comportamentos pseudo-aleatórios, e seleção das respostas mais próximas de resultados desejáveis. A modelagem de um algoritmo evolutivo, como descrito no artigo supracitado é baseada em uma matriz com valores aleatórios, que podem ou não respeitar regras de ordenação obrigatória, limite de valor ou até mesmo tipagem.

"A computação evolutiva deve ser entendida como um conjunto de técnicas e procedimentos genéricos e adaptáveis, a serem aplicados na solução de problemas complexos, para os quais outras técnicas conhecidas são ineficazes ou nem sequer são aplicáveis. " Zuben (2000).

Como descrito, trata-se de um conceito útil para situações em que a programação convencional, buscando resolver o problema por meio de um computador da mesma forma que um ser humano faria manualmente, não se aplica ou falha ao ser aplicada.

Na situação de adequar-se à um comportamento tão humano quanto a sugestão de palavras, seria impraticável o uso de programação convencional, portanto a metodologia escolhida foi o algoritmo (ou software) evolutivo.

Outros conceitos descritos por Zuben são:

- Genes: “Blocos funcionais de DNA, os quais codificam uma proteína específica. É a denominação que damos hoje ao fator mendeliano. ” Na Programação evolutiva, os genes serão valores de tipagem variante, que determinam um comportamento.
- Cromossomo: “estrutura nucleoprotéica formada por uma cadeia de DNA, sendo a base física dos genes nucleares, os quais estão dispostos linearmente. Cada espécie apresenta um número característico de cromossomos. ”
- Crossover: "(recombinação): consiste na troca (evento aleatório) de material genético entre dois cromossomos." Na prática, um software evolutivo utiliza esse conceito mesclando o genoma de dois ou mais indivíduos para formar um terceiro, que terá características de ambos, mas não será idêntico a nenhum.
- Genoma: "como muitos organismos apresentam células com mais de um cromossomo, o genoma é o conjunto de todos os cromossomos que compõem o material genético do organismo." Aplicado à metodologia evolutiva de resolução de problemas, o genoma é tido como uma matriz de cromossomos, com uma ordem e valores bem definidos, definem o comportamento final de um indivíduo.
- Genótipo: "Indivíduos e espécies podem ser vistos como uma dualidade entre seu código genético (genótipo)" Isso é, o código genético em si, não o comportamento determinado por ele.

- Fenótipo: "características comportamentais, fisiológicas e morfológicas (fenótipo)" Isso é, os comportamentos definidos pelos genes, não os genes em si.
- Pleiotropia: "um único gene pode afetar diversos traços fenotípicos simultaneamente (pleiotropia)" Um gene pode determinar dois comportamentos distintos e simultâneos.
- Poligenia: "uma única característica fenotípica pode ser determinada pela interação de vários genes (poligenia)." Vários genes, interagindo entre si, podem determinar uma característica única.
- Fitness: "uma função que mede a adequação relativa de cada indivíduo frente aos demais" A medida dada à adequação do comportamento tido pelo individuo frente ao problema a se resolver, que deve ter como característica uma universalidade, onde dois objetos expostos ao mesmo problema e com o mesmo fenótipo devem ter valores iguais.
- Transformação unária: “Existem transformações unárias (mutação) que criam novos indivíduos através de pequenas modificações de atributos em um indivíduo”. Na prática, é alguma alteração de um gene em um indivíduo, adicionando uma nova possibilidade ao universo de valores a se utilizar.
- Seleção de indivíduos: “O algoritmo genético clássico utiliza um esquema de seleção de indivíduos para a próxima geração chamado roulettewheel”. Essa seleção é o que define quais indivíduos permanecerão para a próxima geração, fazendo com que a evolução aconteça. É o análogo à seleção natural, quase sempre baseando-se no fitness.

No artigo também são exemplificados várias formar de crossover e mutação (Transformação unária). Tais como:

- Crossover de um ponto, onde a matriz genética de dois indivíduos é dividida a partir de um ponto de forma que um novo é formado com a primeira parte do primeiro genoma e a segunda parte do segundo genoma e outro com o inverso, a segunda parte do

primeiro genoma e a primeira parte do segundo genoma. O ponto pode ser fixo, flutuante, ou ainda obedecer algumas regras, dependendo do problema a ser solucionado.

- Crossover uniforme, onde para cada gene é decidido de forma aleatória qual genitor fornecerá o valor.
- Crossover de dois pontos, que como o nome sugere é uma modelagem do crossover de um ponto onde há mais de um ponto de corte no genoma dos genitores.
- crossover OX, que funciona com o mesmo principio do crossover de um ponto, mas com a substituição de valores repetidos pelos ausentes no genoma. Por exemplo, se houverem oito possíveis valores para os genes, mas ao executar o corte um individuo tem o valor ‘um’ repetido e o ‘dois’ ausente (nesse caso haveriam oito genes por indivíduo), o um dos valores ‘um’ seria substituído pelo valor ‘dois’.

O autor ainda complementa: “não há nenhum operador de crossover que claramente apresente um desempenho superior aos demais. Uma conclusão a que se pode chegar é que cada operador de crossover é particularmente eficiente para uma determinada classe de problemas e extremamente ineficiente para outras. ”

Quanto aos operadores de mutação, o artigo também apresenta a descrição de algumas opções:

- “Considerando codificação binária, o operador de mutação padrão simplesmente troca o valor de um gene em um cromossomo”
- “O operador para mutação uniforme seleciona aleatoriamente um componente k □ {1, 2, ..., n} do cromossomo x = [x1 ... xk ... xn] e gera um indivíduo x′ = [x1 ... x′k ... xn] , onde xk′ é um número aleatório” Isso é, ao mutar um indivíduo, esse operador altera um gene aleatório para ter um valor aleatório.

Outro ponto descrito cujas abordagens são múltiplas é a seleção dos indivíduos para a geração seguinte:

- Roulettewheel: “O roulettewheel atribui a cada indivíduo de uma população uma probabilidade de passar para a próxima geração

proporcional ao seu fitness medido, em relação à somatória do fitness de todos os indivíduos da população. Assim, quanto maior o fitness de um indivíduo, maior a probabilidade dele passar para a próxima geração."

- Rank: "A seleção baseada em rank [...] utiliza as posições dos indivíduos quando ordenados de acordo com o fitness para determinar a probabilidade de seleção. Podem ser usados mapeamentos lineares ou não-lineares para determinar a probabilidade de seleção. Para um exemplo de mapeamento não-linear, veja MICHALEWICZ (1996). Uma variação deste mecanismo é simplesmente passar os N melhores indivíduos para a próxima geração."
- Seleção por diversidade: "são selecionados os indivíduos mais diversos da população."
- Seleção bi-classista: "são selecionados os P% melhores indivíduos e os (100 − P)% piores indivíduos."
- Seleção aleatória: "são selecionados aleatoriamente N indivíduos da população. Podemos subdividir este mecanismo de seleção em: Salvacionista: seleciona-se o melhor indivíduo e os outros aleatoriamente. E não-salvacionista: seleciona-se aleatoriamente todos os indivíduos."

"Estes mesmos mecanismos de seleção podem ser adaptados para selecionar também os indivíduos que irão sofrer crossover e mutação. Por exemplo, usando a seleção bi-classista, é possível selecionar os indivíduos que, ao se reproduzirem, irão gerar os indivíduos da próxima geração. "

2.3 O que é autocomplete

Autocomplete de acordo com a ComputerHope no seu artigo sobre autocomplete, é o que faz com que sugestões plausíveis apareçam conforme o usuário digita algo, por exemplo você escreve "c" e então aparecera uma lista com "casa" , "carro" , "casaco a venda", "vende-se carro", ou seja ao escrever "c" sera procurada uma ou mais informações que contenham a letra "c", sempre mostrando as mais utilizadas , comuns e plausíveis.

Isso ajuda o usuário e o deixa mais confortável ao preencher um formulário, ele é importante para deixar o preenchimento desses campos mais dinâmico, mais rápido, afinal todos os usuários sempre estão em busca de um meio para fazer suas pesquisas de forma mais rápida, para poupar tempo tal que sera usado de forma mais produtiva em outro momento.

2.4 Programação Orientada a Objetos

A programação Orientada a Objetos de acordo com Nery na sua abordagem descrita em sua apostila Programação Orientada a Objeto(POO), é um paradigma, modelo, padrão de programação de computadores, que usa de conceitos voltados à Objetos e Classes como elementos centrais para representar e processar os dados usados nos softwares. A ideia da POO é que poderíamos construir um programa usando conceitos e abstrações do mundo real, como objetos.

O princípio da abstração é a capacidade de abstrair a complexidade de um sistema e se concentrar apenas em partes do mesmo, por exemplo," um médico torna-se um especialista em algum órgão do corpo, digamos que seja o coração. Ele abstrai sem desconsiderar as influências dos outros órgãos e foca apenas sua atenção nesse órgão" - Nery.

Os objetos mencionados anteriormente são usados para representar entidades do mundo real ou computacional, se observarmos ao nosso redor, será possível ver várias entidades ou abstrações as quais podem ser representadas como objetos no nosso programa, as escolas e seus alunos podem ser vistas como objetos, os objetos têm dois pontos principais, as características ou seja, atributos pelos quais os identificamos e as finalidades para as quais os utilizamos um objeto pode ter comportamentos associados, chamados de métodos, por exemplo, uma pessoa pode andar, correr ou dirigir, e um carro pode ligar, desligar, acelerar e frear.

Utilizamos de Classes para criar nossos objetos, cada classe funciona como um molde, uma forma para a criação de um dado objeto, portanto os objetos são vistos como representações completas, ou seja, instâncias das classes.

A Programação Orientada a Objetos é sedimentada sobre três pilares que provêm do princípio da abstração, sendo eles:

- Encapsulamento: “deriva-se de cápsula, que nos lembra qualquer forma pequena que protege algo em seu interior como um medicamento. Ele ajuda a desenvolver programas com maior qualidade e flexibilidade para mudanças futuras.” - Nery O encapsulamento também é capaz de ocultar partes(dados e detalhes) de implementação interna de classes do mundo exterior.
- Herança: é o mecanismo que permite a uma classe herdar todos os atributos e métodos de outra classe. “Quanto mais alta a classe na hierarquia, mais ela tende a ser abstrata” - Nery, ou menos definida, com menos atributos e métodos.Suponha que há uma classe Transporte, ela tem o atributo capacidade, ela pode ter uma classe filha chamada Terrestre ou Aquático que tem seus próprios atributos e métodos mais específicos
- Polimorfismo: “deriva-se da palavra polimorfo, que significa multiforme, de várias formas, ou que pode variar a forma. Para a Programação Orientada a Objetos, polimorfismo é a habilidade de objetos de classes diferentes responderem a mesma mensagem de diferentes maneiras.” - Nery Ou seja, várias formas de responder à mesma mensagem.Por exemplo, o dono de uma fábrica de brinquedos instruiu seus engenheiros a criar um mesmo controle remoto para todos os brinquedos de sua fábrica, a única restrição era que cada brinquedo atendesse aos comandos específicos definidos pelo controle.

2.5 Java

De acordo com Peter Jr. em Introdução ao Java, a linguagem de programação Java em conjunto de sua plataforma, constituem um fascinante objeto de estudo, com um conjunto rico de bibliotecas para facilitar o desenvolvimento e utiliza como paradigma central a Orientação a Objetos.Javaéra uma nova linguagem de programação no mercado em 1995 pela Sun Microsystems, que provocou e ainda provoca excitação e entusiasmo em programadores, analistas e projetistas de software.

O Java produz essa reação simplesmente porque é o resultado de um trabalho consistente de pesquisa e desenvolvimento de mais do que uma simples linguagem de programação, mas de todo um ambiente de desenvolvimento e execução de programas que exibe as finalidades proporcionadas pela orientação à objetos, pela extrema portabilidade do código produzido, pelas características de segurança que esta plataforma oferece e finalmente pela facilidade de sua integração aos outros ambientes, destacando-se a Web.

Algumas de suas características mais importantes de acordo com Peter Jr. que, agrupadas, diferenciam-na de outras linguagens de programação são:

- Orientado à Objetos: com exceção de seus tipos primitivos de dados, tudo em Java são classes ou instâncias delas.Java atende todos os requisitos necessários para uma linguagem ser considerada orientada a objetos, que resumidamente são oferecer mecanismos de abstração, encapsulamento e hereditariedade.
- Independente de Plataforma: “todo o código Java é compilado para uma forma intermediária de código denominada bytecodes que utiliza instruções e tipos primitivos de tamanho fixo, ordenação bit-endian e uma biblioteca de classes padronizada. Os bytecodes são como uma linguagem de máquina destinada a uma única plataforma, a máquina virtual Java(JVM), um interpretador de bytecodes. Java pode ser executado em qualquer arquitetura que disponha de JVM.” – Peter Jr.
- Sem Ponteiros: isto é, Java não permite a manipulação direta de endereços de memória nem exige que os objetos criados sejam destruídos livrando os programadores de uma tarefa complexa. Além disso a JVM possui um mecanismo automático para o gerenciamento da memória o garbagecollector, que recupera a memória alocada para objetos mais referenciados pelo programa.
- Performance: “Java foi projetada para ser compacta, independente de plataforma e para utilização de rede o que levou a decisão de ser interpretada através do esquema de bytecodes. Como Java é uma linguagem interpretada a performance é razoável, não podendo ser comparada a velocidade de execução de código nativo. Para superar

esta limitação várias JVM dispõem de compiladores Just In Time(JIT) que compilam os bytecodes para código nativo durante a execução otimizando a mesma, que nestes casos melhora significativamente a performance dos Softwares feitos com Java.”- Peter Jr.

- Segurança: “Considerando, a possibilidade de aplicações obtidas através de uma rede, a linguagem Java possui mecanismos de segurança que podem, no caso de applets, evitar qualquer operação no sistema de arquivos da máquina-alvo, minimizando problemas de segurança.Isso é flexível o suficiente para determinar se a aplicação é segura especificando nessa situação diferentes níveis de acesso ao sistema-alvo” – Peter Jr.
- Multithreading: Java oferece meios para o desenvolvimento de softwares capazes de executar múltiplas rotinas concorrentemente, também dispõe de elementos para a sincronização das mesmas. Cada um destes fluxos de execução é o que se denomina thread, um importante recurso de aplicações mais sofisticadas.

Java é uma linguagem muito completa, oferece tipos com as especificações IEEE, suporte para caracteres UNICODE, é extensível dinamicamente, além de ser voltada por natureza ao desenvolvimento de aplicações em rede ou distribuídas.

2.6 UML

SegundoGudwin, a linguagem UML(UnifiedModelingLanguage) é uma linguagem de modelagem criada visando-se a criação de modelos abstratos de processos sendo modelados. Tanto podem ser processos do mundo real, como processos de desenvolvimento de software ou ainda detalhes internos do próprio software. Assim tanto podemos utilizá-lo para descrever o mundo real, como a organização de uma empresa, como os detalhes internos que descrevem um sistema de software. A descrição de um processo envolve a determinação de duas classes básicas de termos:

- Os elementos estruturais que compõem o processo;
- O comportamento que esses elementos desenvolvem quando interagindo.

“Uma das características interessantes do UML é a existência de mecanismos de extensão, que permitem que o UML, como linguagem, possa ser estendido, resultando a criação de novos tipos de diagramas. Os mecanismos de extensão do UML são os chamados Profiles. ”- Gudwin. Diferentes Profiles podem ser construídos utilizando-se estereótipos, os taggedvalues e as restrições.

“A linguagem UML, por meio de seus diagramas, permite a definição e design de threads e processos, que permitem o desenvolvimento de sistemas distribuídos ou de programação concorrente. “- Gudwin. Da mesma maneira, permite a utilização dos chamados patterns (patterns são, a grosso modo, soluções de programação que são reutilizadas devido ao seu bom desempenho) e a descrição de colaborações (esquemas de interação entre objetos que resultam em um comportamento do sistema).

“Um dos tipos de diagramas particularmente úteis para modelarmos processos são os chamados diagramas de atividades. Por meio deles, é possível especificarmos uma sequência de procedimentos que compõem um processo no mundo real. ”- Gudwin. Diagramas de atividade podem, portanto, ser utilizados para descrever o processo de desenvolvimento de software (por exemplo, o processo Unificado).

Uma outra característica do UML é a possibilidade de efetuarmos uma descrição hierárquica dos processos.

Com isso, é possível fazermos um refinamento de nosso modelo, descrevendo o relacionamento entre diferentes níveis de abstração do mesmo.

Para implementar esse refinamento, utilizamos o expediente de definir abstratamente componentes de um determinado modelo por meio de suas interfaces. Assim, esses componentes são definidos somente em função de suas entradas e saídas, deixando a definição dos internals do componente para um nível de abstração posterior. Essa característica permite ao UML um desenvolvimento incremental do modelo.

Por fim, a semântica dos diagramas UML é determinada por uma linguagem de restrição chamada de OCL (ObjectContraintLanguage), que determina de maneira não-ambígua a interpretação a ser dada a seus diagramas

2.7 HTML

HTML-HiperTextMarkupLanguageEis(2011), é uma linguagem de marcação usada como base para sites web, com ela nós marcamos elementos para mostrar quais informações a página exibe, exemplo do uso da tag<h1>:

<h1>Título aqui!</h1>

Como se pode ver há um texto entre duas marcações, as marcações <h1> não vão aparecer na página mas vão modificar o texto que há entre elas.

Para cada marcação de abertura sempre há uma de fechamento, no caso para será . Também há outras marcações como <h1> para títulos.

Uma das principais características do HTML se reflete no fato de se um programa navegador não "entender" um determinado comando, este é ignorado e não *é* apresentado, não originando mensagem de erro e afetando minimamente o restante do documento.

Assim, resumidamente, quando se digita um endereço de um site, o navegador

1. Contata o servidor de DNS e descobre onde está o computador que hospeda o site desejado;
2. Envia o pedido de cópia do(s) arquivo(s) que está naquele endereço;
3. Então, o servidor web analisa e trata o pedido e responde ao navegador com um arquivo de texto;
4. O navegador obedece o texto e constrói a página na tela do cliente;
5. A pessoa *vê*, em seu monitor, a página web solicitada;

“Entretanto, como o HTML é uma linguagem descritiva, de formatação, nem sempre diferentes navegadores exibem a mesma apresentação em cada página. Ou seja, os detalhes codificados no HTML podem ser suficientes para um deles mas não suficientes para outro.” – Eis.

Assim, cada navegador poderá interpretar os dados de uma forma um pouco diferente. Portanto, quanto mais perfeitamente descrita a página for, maiores serão as chances do documento ser interpretado da mesma forma por diversos navegadores.

O HTML não foi criado para controlar a aparência das paginas, ele apenas indica ao navegador o que é o conteúdo da pagina, quais arquivos ela contém e onde estão usando as marcações.

Porém o HTML é estatico, sem movimento ou qualquer outra coisa qe atraia a atenção do usuário, mas ele pode ser dinamizado por linguagens como JavaScript e CSS.

“Assim, HTTP é o protocolo usado pelos computadores para a transmissão de dados na World WideWeb(Exemplo : “www.site.dominio”).” – Eis.

É importante lembrar que hipertexto é um texto que pode apresentar sons, vídeos, imagens e outras aplicações. Ou seja, os dados que podem ser transferidos podem ser de qualquer tipo.

Porém, o que realmente caracteriza o hipertexto é que pode se criar links, isto é, ligações para outros arquivos (textos, imagens, sons, vídeos). Assim, os links possibilitam a "navegação", tanto dentro de um arquivo,site , de um ponto para outro, *ou* entre arquivos diferentes, que podem estar em computadores também diferentes, e que podem estar localizados proximamente ou estar extremamente distantes no nosso plano terrestre

“Assim quando se clica em um link ele faz toda a comunicação com o servidor novamente, a fim de alcançar o arquivo no final do caminho que o link está a apontar. ” –Eis.

2.8 CSS

De acordo com Guimarães na sua apostila de introdução ao CSS, CascadingStyleSheets(CSS) é usado para garantir uma formatação homogenia

e uniforme nas páginas de um website, ou seja, CSS é um padrão de formatação para páginas, o que nos permite sair das limitações de layout e legibilidade de um arquivo HTML.

O CSS possui uma regra simples que pode ser dividida em duas partes, seletor e declaração.

O seletor, como o próprio nome já o denuncia, é o que liga elemento à declaração.

A declaração possui duas subpartes, propriedade e valor.Propriedade é uma qualidade ou uma característica que um elemento deve possuir.

Mais a fundo há três mantras que devem ser seguidos ao se aplicar o CSS nas páginas Web, que são respectivamente:

- Inline: em linha, diretamente no elemento que você deseja afetar, você pode chamar uma propriedade style e escrever CSS que será atribuído à propriedade;
- Embedding: interno, incorporado, embutido, nesse caso você usa uma marcação de texto do HTML(<style>) e dentro dela você pode escrever livremente seus seletores com suas declarações, que afetarão somente o arquivo em que estão sendo escritas;
- Linking: externo, esse é o mantra usado por programadores experientes, pois permite a reutilização dos estilos para múltiplas páginas sem poluir visualmente seu texto HTML.Essa referência ao arquivo CSS é feita a partir de uma marcação de texto do html(<link>), nele você insere o caminho a partir do arquivo em que você está desenvolvendo está situado no servidor, computador, etc...

2.9 IDE

IDE, ou Ambiente Integral de Desenvolvimento em tradução livre descrito por Novaes, é um software criado com a finalidade de facilitar a vida dos programadores. Neste tipo de aplicação estão todas as funções necessárias para o desenvolvimento desde programas de computador a aplicativos mobile, assim como alguns recursos que diminuem a ocorrência de erros nas linhas de código.

“Se no passado os desenvolvedores precisavam apenas de um editor de texto e de um navegador para criar um software, agora, com os IDEs, eles possuem mais opções para otimizar o tempo gasto com os códigos. Imagine os IDEs como as calculadoras. Logicamente você aprende a fazer as operações matemáticas na escola, mas raramente as faz manualmente quando precisa.” – Novaes.

Uma das principais vantagens dos IDEs está na capacidade de compilar bibliotecas completas de linguagem. Outra função bastante comum neste tipo de programa são os debuggers, que apontam os erros que ocasionalmente podem ocorrer ao escrever o código. Alguns IDEs também possuem o autocomplete.

Já a desvantagem fica por conta de necessitar um conhecimento razoável de programação. Usuários com pouca experiência – ou que estão dando os primeiros passos no desenvolvimento de software – podem se confundir com o excesso de recursos que alguns IDEs têm.

2.10 Banco de Dados

Segundo DATE(2004, p. 10), “Um banco de dados é uma coleção de dados persistentes, usada pelos sistemas de aplicação de uma determinada empresa”.Em outras palavras, um banco de dados é um local onde são armazenados dados necessários à manutenção das atividades de determinada organização, sendo este repositório a fonte de dados para as aplicações atuais e as que vierem a existir.

Para ELMASRI e NAVATHE (2011, p. 3), na expressão Banco de Dados estão subentendidas as propriedades abaixo:

> “Um banco de dados representa algum aspecto do mundo real, às vezes chamado de minimundo ou de universo de discurso (UoD – UniverseofDiscourse). As mudanças no minimundo são refletidas no Banco de Dados. Um banco de dados e uma coleção logicamente coerente de dados com algum significado inerente. Uma variedade aleatória de dados não pode ser corretamente chamada de banco de dados. Um banco de dados é projetado, construído é populado com dados para uma finalidade específica. Ele possui um grupo definido

> de usuários e algumas aplicações previamente concebidas nas quais esses usuários estão interessados.”

Portanto um banco de dados nada mais é do que um conjunto de dados relacionados, criado com dado objetivo que atende uma comunidade de usuários.

2.10.1 MER

De acordo com O. K. Takai, I.C. Italiano e J. E. Ferreira(Introdução a Banco de Dados p. 22), MER é um modelo de dados conceitual de alto nível, ou seja, seus conceitos foram feitos de forma que praticamente qualquer pessoa possa compreender o que se passa em um. Atualmente sendo usado durante a projeção do banco de dados.

O MER dispõe de alguns conceitos básicos para sua compreensão:

- Entidades: são representações de coisas do mundo real independentes, podem ser pessoas, empresas, escolas, etc… Cada Entidade possui características particulares, chamadas de Atributos.
- Atributos: são as características que descrevem uma entidade, por exemplo, a entidade aluno tem nome, rg, cpf, idade, ano, etc…
- Relacionamento: é uma estrutura que indica uma associação entre instâncias de duas ou mais entidades.

[ALGUMA IMAGEM PRA ISSO]

- Relacionamento Binário: é a relação de dois conjuntos de entidades distintos.

[OUTRA IMG]

- Relacionamento Ternário: é a relação de três conjuntos de entidades distintos.

[OUTRA IMG]

- Cardinalidade: Indica o número de instâncias de entidades que podem estar associadas umas as outras através de um relacionamento, por exemplo, um aluno pode estar em várias escolas, e uma escola pode ter vários alunos, ou seja , é uma cardinalidade de N para N.

[OUTRA IMG]

- Cardinalidade 1:1 : Uma entidade no grupo A está associada no máximo a uma entidade no grupo B e uma entidade em B está associada no máximo a uma entidade em A, por exemplo, uma cadeira tem espaço pra 1 pessoa, enquanto 1 pessoa só pode se sentar em uma cadeira.

[OUTRA IMG]

- Cardinalidade 1:n : Uma entidade em A está associada a qualquer número de entidades em B, enquanto uma entidade em B está associada no máximo a uma entidade em A, por exemplo um cartão de crédito pode ter apenas 1 dono enquanto uma pessoa pode ter N cartões de crédito.

[OUTRA IMG]

- Cardinalidade n:n : uma entidade em A está associada a qualquer número de entidades em B e vice-versa.

[OUTRA IMG]

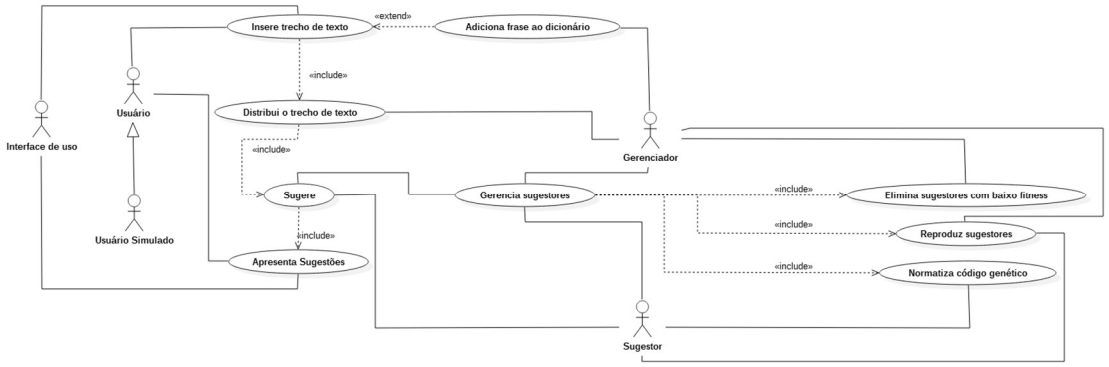
Portanto com o MER podemos montar nosso banco de dados com base em uma regra de negócios.

3 DESENVOLVIMENTO

3.1 Escopo

O desenvolvimento de um software inteligente, ainda mais no âmbito de um algoritmo evolutivo depende, inicialmente, na delimitação de um escopo. O intuito do protótipo desenvolvido é explorar as possibilidades de uso do software evolutivo em um auto-complete, sem a obrigação de ser bem-sucedido nesse processo.

A primeira definição de escopo feita foi um diagrama de casos de uso, que considera alguns atores, tais como o usuário, o usuário simulado, as diferentes entidades com soluções propostas ao problema, e gerenciadores, da interface apresentada e das próprias entidades que sugerem.



É da responsabilidade do usuário (e do usuário simulado) inserir texto base, e escolher uma ou nenhuma das sugestões apresentadas. Essas ações se dão através da interface de uso, que gerencia esses processos e apenas isso.

O gerenciador, um objeto que contém e controla todas as entidades do software evolutivo, doravante chamadas de sugestores, é responsável por:

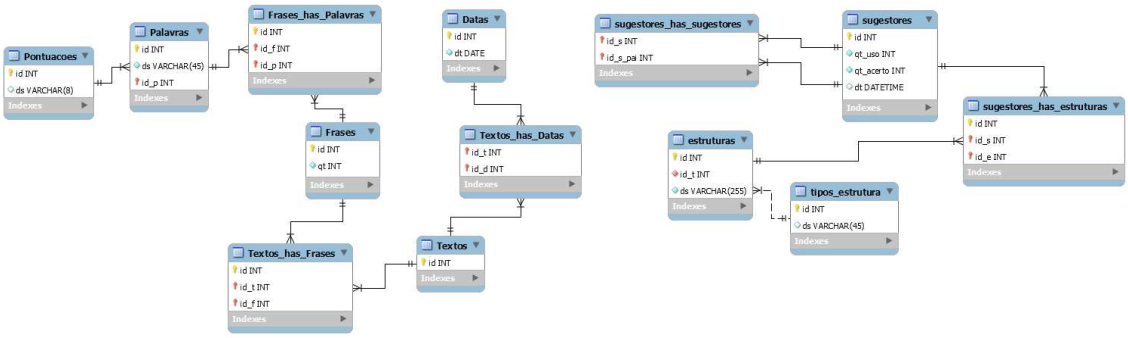
- Inserir os textos no banco de dados, construindo uma biblioteca de referências para os sugestores.
- Gerenciar (eliminar, reproduzir, executar, normalizar o código e mutar) os sugestores.

Os sugestores por si só, são responsáveis por gerar uma alternativa possível de continuidade no trecho de texto digitado pelo usuário.

Todos os casos de uso receberam diagramas de atividades, os mais importantes seguem:

// TODO

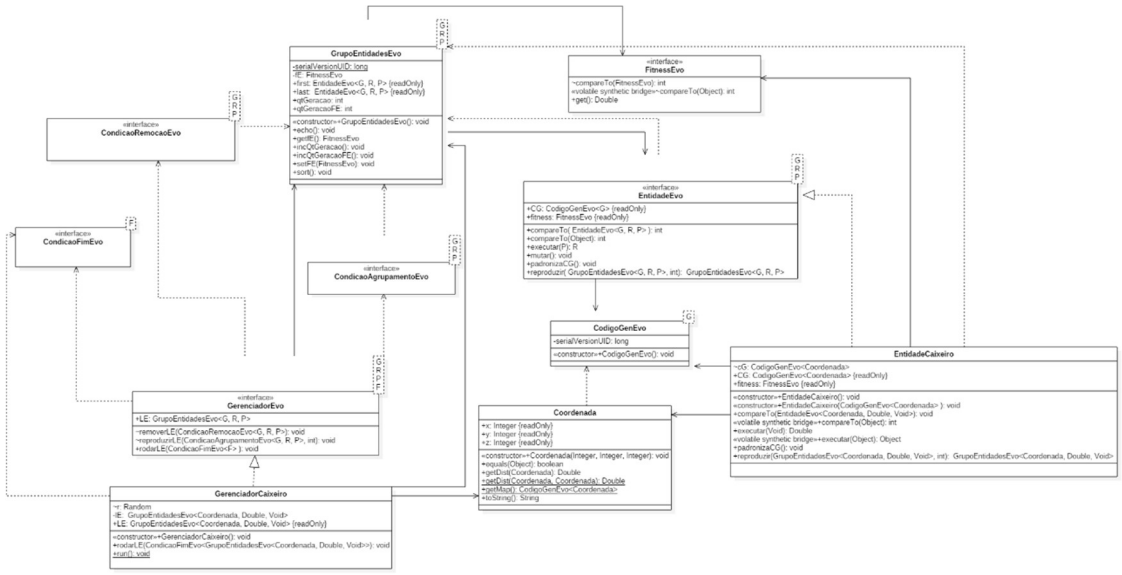
Também foi definido, em termos genéricos, o modelo entidade relacionamento do banco de dados. Com divisão de palavras, frases, textos, sugestores e estruturas.



Mas essas definições não são, no entanto, o bastante para o desenvolvimento do protótipo.

3.2 Bibliotecas

Foram construídas, na linguagem Java, uma série de classes com a função de tratar no escopo do software evolutivo, todos os problemas associados. Sem, não obstante, amarrar-se a soluções delimitadas, isso é, permitindo seu re-uso.



Essa biblioteca organiza-se com base em dois componentes, o GerenciadorEvo e a EntidadeEvo. O primeiro define alguns métodos, como o que executa cada uma das entidades contidas, o que as reproduz, agrupando-as conforme uma regra recebida, e o que remove as entidades com base numa regra. O componente EntidadeEvo tem alguns métodos, responsável por mutar, normalizar e mesclar o código genético e utilizá-lo em uma execução.

Esses tratamentos são tão genéricos quanto possível. Como por exemplo o ciclo de reprodução dessas entidades. O gerenciador recebe uma regra para fazer o agrupamento, e conforme essa regra, ele envia à primeira entidade as listas. A entidade por sua vez não define como essas listas gerarão novos indivíduos, mas possui um método abstrato, isso é, essa definição é feita em uma classe filha.

Essa biblioteca segue os padrões definidos por Zuben (2000), com alguns acréscimos:

- A tipagem do código genético, descrita por Zuben como sendo sempre booleana ou de ponto flutuante perde essa limitação, isso é, podendo receber qualquer coisa, desde um inteiro à uma classe criada pelo programador.
- Tal como a tipagem, a quantidade de alelos para o código genético de cada entidade não recebe uma limitação interna à biblioteca, sendo diferençável para cada problema.

- Na execução das entidades, elas podem receber parâmetros, fazendo que um mesmo código genético possa ter comportamentos diferentes em diferentes contextos, o que é bastante importante para o problema proposto.
- A própria normalização do código genético, com o intuito de evitar erros, recebe aqui uma importância maior do que a descrita por Zuben.

Essa biblioteca também é dotada de outra característica a ser descrita, ela foi feita tão orientada a objetos quanto possível. Evitamos ao máximo linhas de código repetidas e, por diversas vezes, criando interfaces que não propriamente assinavam métodos.

Por exemplo, ao invés de utilizar a interface do Java `Function<List<EntidadeEvo<>>, List<List<EntidadeEvo<>>>>` nós criamos uma interface `AgrupaEntidades` que a estende. Nesse caso, a Function representa a regra de agrupamento de genitores, agrupando quais entidades terão o código genético mesclado para geração de novas.

Essa parcela do software, pode-se chamar de biblioteca, foi desenvolvida em conjunto com uma aplicação no intuito de resolver o problema do caixeiro viajante. No intuito de encontrar as falhas que a análise inicial acarretaria.

O problema do caixeiro viajante é bastante antigo, e cabe aqui uma breve descrição: Um caixeiro viajante quer passar por uma dada quantidade de cidades, ele consegue ir de qualquer uma para qualquer outra, sem limitações geográficas, mas ele quer fazê-lo de forma que passe uma vez por cada cidade, retornando à primeira no final, e de forma que, a soma das distâncias percorridas seja a menor possível.

É um problema de resolução aparentemente simples, considerando seis ou sete cidades, em um plano bidimensional, mas conforme aumentamos a quantia de cidades, a resolução fica simplesmente impossível. Imagine cem cidades, a quantia total de possibilidades de rotas é noventa e nove em fatorial. Ou imagine se fosse em um plano com três dimensões. Torna-se simplesmente impossível para um ser-humano achar a solução mais otimizada.

A aplicação da biblioteca na resolução desse problema deu resultados bastante otimizados, em mais da metade das tentativas retornando uma solução aceitável, e em mais de vinte por cento dos casos sendo a solução mais otimizada que pudemos encontrar.

Há uma questão a ser explicada quanto aos resultados. Uma vez que algoritmos evolutivos são uma forma de busca guiada, com características randômicas, os resultados podem variar, da solução perfeita, dita a mais otimizada, à soluções não tão boas, ditas pouco otimizadas.

Foram utilizadas, conforme no exemplo descrito por Zuben, as coordenadas como tipagem do código genético. A normalização serviu, nesse caso, para que ele não repetisse nenhuma cidade. A mutação foi feita trocando cidades de posição no código genético, ou blocos de cidades. A reprodução era feita dando preferência aos com maior fitness. O primeiro teria um contra dois de passar seu código genético, se não for ele, o próximo terá a mesma chance, depois o terceiro, e se o último falhar, volta-se ao primeiro. Obviamente, o fitness é medido pela menor soma das distâncias.

Uma vez definido que as estruturas funcionam, segue-se para o desenvolvimento de outro teste. Baseando-se nas mínimas partes divisíveis de um algoritmo, a intenção era a construção de uma sub biblioteca para as estruturas programadas do software evolutivo. A tipagem do código genético seria uma interface chamada SubAlgoritmo que seria implementada por estruturas de programação, tais como: Estrutura condicional, estruturas de repetição, estruturas de contas matemática, comparação, e de operações lógicas.

A implementação foi relativamente bem-sucedida, uma vez resolvido o problema da normalização, que precisava garantir que todas as estruturas recebessem os parâmetros necessários (Uma soma precisa receber pelo menos dois números). Mas não fomos além disso, a aplicação do conceito de software evolutivo na construção de algoritmos sofre dificuldades severas quando se trata de verificar o fitness, de reproduzir as entidades ou de causar mutações nelas.

Qualquer alteração faz com que o funcionamento se altere completamente. E não é possível um crossover eficiente, no que se trata de

aproveitar qualidades positivas dos genitores. Pelo o menos não reaproveitando qualquer estrutura de crossover descrita por Zuben.

A intenção inicial era uma aplicação semelhante a relativa aos sub algoritmos na construção de comandos SQL de seleção, para as palavras. Mas os mesmos problemas seriam aplicáveis, não se pode fazer qualquer mudança real em uma seleção sem alterar completamente o resultado, perdendo completamente quaisquer qualidades que se pudesse ter anteriormente, em decorrência de todos os genes, nessa análise, se referirem à uma mesma característica. Também não se pode mesclar duas seleções com simplicidade, as características vantajosas seriam perdidas com qualquer mistura.

Nesse caso, seria mais efetivo uma análise mais direcionada ao problema, perdendo a capacidade de adaptar o que se desenvolveu na resolução de outros problemas.

Essa análise mais direcionada poderia se dar com a construção de uma entidade evolutiva com mais de um método, que executaria diferencialmente, mas cada um tendo uma finalidade simples e bem definida. E resolver cada método com um único objeto, que seria a tipagem genética. Por exemplo, se a intenção fosse a construção de entidades para servirem de inimigos do usuário em um jogo. Para cada funcionalidade que o inimigo pudesse executar, haveria um gene responsável. Um que o faria se mover, um que o faria atirar, e assim por diante.

Nessa análise, um gene poderia inclusive utilizar do funcionamento de outro, como quando mover o boneco quando ele atira. Isso poderia produzir efeitos desejáveis, mas exigiria um grande cadastro de possibilidades de resolução para cada pequeno problema. Seria preciso vários tipos de movimentação e de tiro, no exemplo do jogo.

3.3 Implementação

A análise final, dada ao problema, foi a opção mais direcionada. A implementação dada à entidade tem como tipagem genética estruturas de seleção SQL, à serem cadastradas com dada escala, sem limitação declarada para a quantidade de genes.

A execução dessas entidades dá-se enviando a concatenação das estruturas para o banco, em forma de seleção. O resultado considerado é sempre o primeiro, de forma que a ordenação de interna cada gene (o order by do SQL) é extremamente impactante, bem como a ordenação das diferentes estruturas dentro da entidade.

A princípio, essa análise não valoriza muito a utilização do software evolutivo em si, uma vez que a quantidade se estruturas cadastradas será limitada. Isso é, seria possível para o programador escolher quais dessas estruturas são mais eficientes e concatena-las uma vez só. Seria o equivalente, no problema do caixeiro viajante, à um ser humano olhar no mapa e verificar a rota mais rápida.

No entanto essa análise ainda preserva algumas características desejáveis por utilizar algoritmos evolutivos. Como a adaptação à erros comuns de dado usuário, como por exemplo, adicionar, frequentemente, uma letra a mais, ou errar uma tecla.

Ainda que essas características adaptativas advindas da aplicação evolutiva na busca da melhor solução possam ser, potencialmente, resolvidas com o desenvolvimento convencional de softwares, alcança-se os objetivos estipulados no início do desenvolvimento desse trabalho.

4 CONCLUSÃO

Em decorrência do exposto, pode-se concluir quebláblábláblábláblábláblábláblábláblábláblábláblábláblábláblábláblábláblábláblá bláblábláblábláblábláblá bláblábla bláblábláblábláblábláblábláblábláblábláblábláblábláblábláblábláblábláblábláblá bláblábla. Blábláblábláblábláblábláblá bláblábla blábláblábláblá bláblábla bláblábláblábláblábláblábláblábláblábláblá bláblábla blábláblábláblábláblábláblábláblábláblábláblábláblábláblábláblábláblábláblábláblab lábláblábla bláblábla.

Blábláblábláblábláblábláblá bláblábla blábláblábláblá bláblábla bláblábláblábláblábláblábláblábláblábláblá bláblábla blábláblábláblábláblábláblábláblábláblábláblábláblábláblábláblábláblábláblábláblab lábláblábla bláblábla. Blábláblábláblábláblábláblá bláblábla blábláblábláblá bláblábla bláblábláblábláblábláblábláblábláblábláblá bláblábla blábláblábláblábláblábláblábláblábláblábláblábláblábláblábláblábláblábláblábláblab lábláblábla bláblábla.

Blábláblábláblábláblábláblá bláblábla blábláblábláblá bláblábla bláblábláblábláblábláblábláblábláblábláblá bláblábla blábláblábláblábláblábláblábláblábláblábláblábláblábláblábláblábláblábláblábláblab lábláblábla bláblábla. Blábláblábláblábláblábláblá bláblábla blábláblábláblá bláblábla bláblábláblábláblábláblábláblábláblábláblá bláblábla blábláblábláblábláblábláblábláblábláblábláblábláblábláblábláblábláblábláblábláblab lábláblábla bláblábla.

5 REFERÊNCIAS

BRASIL Tribunal de Justiça do Estado de São Paulo (2. Câmara). Civil. A ementa pode ser acrescentada aqui. Agravo de Instrumento nº 243.762-1. Relator: Desembargador J. Roberto Bedran. São Paulo, 7 de fevereiro de 1995. Revista Jurídica, Porto Alegre. Disponível em: <http:/www.jol.com.br>. Acesso em: 19 mar. 1998.

BRASIL. Tribunal de Justiça do Estado do Rio Grande do Sul (6. Câmara). Civil. A ementa pode ser acrescentada aqui. Apelação Cível nº 596076380. Relator: Desembargador Paulo Roberto Hanke. Porto Alegre, 25 de junho de 1996. Revista Jurídica, Porto Alegre. Disponível em: <http:/www.jol.com.br>. Acesso em: 19 mar. 1998.

CAMARGO, Margarida Maria Lacombe. Direito e hermenêutica. *Revista Ciências Sociais*, Rio de Janeiro, v. 4, n. 1, p. 188-213, jun. 1998.

KELSEN, Hans.*O que é justiça?*: a justiça, o direito e a política no espelho da ciência. Tradução Luís Carlos Borges. 3. ed. São Paulo: M. Fontes, 2001. 404 p. (Justiça e Direito). Tradução de: Whatis justice?

KELSEN, Hans. *Teoria pura do direito*. 6. ed. 4. tir. Tradução João Baptista Machado. São Paulo: M. Fontes, 2000. 427 p. (Ensino Superior). Tradução de: Reine Rechtslehre.

LIMA, Hermes. *Introdução à ciência do direito*. 27. ed. Rio de Janeiro: Freitas Bastos, 1983.

NADER, Paulo. *Introdução ao estudo do direito*. 24. ed. rev. e atual. de acordo com o novo Código Civil, Lei nº 10.406, de 10 de janeiro de 2002. Rio de Janeiro: Forense, 2004.

PERELMAN, Chaïm. *Ética e direito*. Tradução Maria Ermantina Galvão. São Paulo: M. Fontes, 1999. 722 p. Título original: Éthique et droit.

PERELMAN, Chaïm; Olbrechts-Tyteca, Lucie. *Tratado da argumentação*: a nova retórica. São Paulo: Martins Fontes, 1996.

RAWLS, John. *Uma teoria da justiça*. Tradução por: Almiro Pisetta e Lenita M. R. Esteves. São Paulo: M. Fontes, 1997. 708 p.

REALE, Miguel. *Lições preliminares de direito*. 27. ed. ajustada ao novo Código Civil. São Paulo: Saraiva, 2002. 391 p.