UNIVERSIDADE ESTADUAL DO MARANHÃO
CENTRO DE CIÊNCIAS EXTAS
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA DE COMPUTAÇÃO E SISTEMAS

IDOVALDO CUNHA DA SILVA

UM MODELO DE SISTEMA TUTOR INTELIGENTE PARA O ENSINO NO DOMÍNIO DE LÓGICA DE PROGRAMAÇÃO

SÃO LUIS/MA
2016

UNIVERSIDADE ESTADUAL DO MARANHÃO
CENTRO DE CIÊNCIAS EXTAS
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA DE COMPUTAÇÃO E SISTEMAS

IDOVALDO CUNHA DA SILVA

UM MODELO DE SISTEMA TUTOR INTELIGENTE PARA O ENSINO NO DOMÍNIO DE LÓGICA DE PROGRAMAÇÃO

Dissertação apresentada ao Programa de Pós-Graduação em Engenharia de Computação e Sistemas - PECS (Mestrado Profissional), da Universidade Estadual do Maranhão – UEMA, como pré-requisito parcial para a obtenção do Título de Mestre em Engenharia de Computação e Sistemas.

Orientador: Prof. Dr. Luís Carlos Costa Fonseca

Co-orientador: Prof. Ms. Reinaldo de Jesus Silva

SÃO LUIS/MA
2016

UM MODELO DE SISTEMA TUTOR INTELIGENTE PARA O ENSINO NO DOMÍNIO DE LÓGICA DE PROGRAMAÇÃO

Esta Dissertação foi julgada adequada para a obtenção do título de Mestre em Engenharia de Computação e Sistemas, Área de Concentração em Informática na Educação, e aprovada em sua forma final pelo Programa de Pós-Graduação em Engenharia de Computação e Sistemas.

Banca Examinadora

_____

Prof. Dr. Luis Carlos Costa Fonseca

Presidente da Banca (Orientador)

_____

Prof. Dr. Angelo Rodrigo Bianchini

Membro da Banca

_____

Prof. Ms. Reinaldo de Jesus Silva

Membro da Banca

“Ninguém ensina nada a ninguém,

no máximo ajuda-se o outro a aprender”

Galileu

Aos meus pais, Domingas Marim e Adão Pereira.

AGRADECIMENTOS

Em primeiro lugar, gostaria de expressar a minha mais profunda gratidão ao Prof. Dr. Luís Carlos Costa Fonseca, meu orientador, por sua inspiração e orientação não somente pela contribuição em minha pesquisa, mas também em outros aspectos da vida. Eu também gostaria de agradecer ao Prof. Ms. Reinaldo de Jesus Silva, meu co-orientador, por seu apoio e críticas construtivas ao longo de minha jornada como mestrando e no processo de desenvolvimento do meu trabalho.

Muitos agradecimentos aos meus amigos Raimunda Nonata, Lanylldo Araujo, Francisco da Conceição, Jardel Almeida e Edênia Coqueiro e aos demais, que por não estarem citados aqui, deixam de ser menos importantes. Meus agradecimentos são também extensivos aos meus ex-professores e atuais colegas, Prof. Dr. Rodrigo Bianchini e Prof. Dr Acildo Leite, e colegas de mestrado que contribuíram significativamente para meu crescimento durante todo o processo como mestrando.

Gostaria de agradecer, também, a Jéssica Aguiar, por estar presente em minha vida durante esses dois anos como mestrando. Obrigado por tudo e pela forma que você transformou na minha vida. Obrigado pelo teu carinho, tua alegria, tua atenção, tua vibração com as minhas conquistas e teu ombro em cada momento difícil que você ajudou a atravessar. Sem você, essa conquista não teria o mesmo gosto.

Por último, mas sempre em primeiro lugar em meus pensamentos, agradeço aos meus avós, Avelina Cunha e Santana Marim (em memória), a meus pais, Domingas Marim e Adão Pereira, pelo apoio incondicional e incentivo, e meus queridos irmãos, Irlane Cunha, Idelane Cunha, Irilane Cunha, Vanderlene Barroso, Idonaldo Cunha, Iveraldo Cunha e Inivaldo Cunha pelo o apoio e confiança em mim depositado.

RESUMO

Ensinar programação introdutória é um desafio à educadores por décadas. Dos métodos sugeridos para melhorar o processo de ensino, a tutoria indivíduo tem demonstrado mais eficácia, mesmo diante de uma grande quantidade de recursos que requisita, sua amplitude, com um grande número de estudantes que querem aprender programação. Uma alternativa viável é a utilização de Sistemas Tutores Inteligentes (STI) para esta finalidade. Embora alguns STI's tenham sidos construídos para ensinar programação, nenhum foi desenvolvido para abordar o tema de lógica de programação fazendo uso de uma linguagem visual, que está se tornando cada vez mais popular para ensino iniciantes os primeiros passos na área de domínio de programação. Esta dissertação trata-se da construção de Um Modelo Sistema Tutor Inteligente através da concepção e construção de um protótipo de um Sistema Tutor Inteligente para preencher essa lacuna no ensinar Lógica de programação. Tem como objetivo apresentar um modelo computacional de Um Sistema Tutor Inteligente (STI), para dar suporte ao processo de ensino e aprendizagem no domínio de lógica de programação utilizando como técnica de Inteligência Artificial - IA, o Raciocínio Baseado em Casos (*Case-based Reasoning* - CBR), bem como fazendo uso de uma base de conhecimento do modelo do domínio. Abordando a concepção, modelagem e prototipação de um Sistema Tutor Inteligente para ensinar lógica de programação a partir do uso de uma linguagem visual. A investigação parte da hipótese de que o protótipo concluído poderá provar que o conceito de STI BLOP (BLOP - Bloco de Lógica de Programação), uma vez desenvolvido, fornecerá um ambiente de aprendizagem interativo e rico para os alunos em seu processo de aprendizagem.

Palavras chaves: Lógica de programação. Algoritmos. Sistema Tutores Inteligentes. Raciocínio Baseado em Casos.

ABSTRACT

Teach introductory programming is a challenge to educators for decades. The methods suggested to improve the teaching process, individual tutoring has proven more effective, even with a lot of features that order, its amplitude, with a large number of students who want to learn programming. A viable alternative is the use of Intelligent Tutoring Systems (ITS) for this purpose. Although some STI's have solid built to teach programming, none have been developed to address the programming logic theme making use of a visual language that is becoming increasingly popular for teaching beginners the first steps in programming domain area. This dissertation deals with the construction of a Model System Intelligent Tutor through the design and construction of a prototype of an Intelligent Tutor System to fill the gap in teaching programming logic. It aims to present a computational model of An Intelligent Tutor System (ITS) to support teaching and learning in the programming logic domain using as Artificial Intelligence technique - IA, the Case Based Reasoning (Case-based Reasoning - CBR) and making use of a domain model of the knowledge base. Addressing the design, modeling and prototyping of an Intelligent Tutor System to teach programming logic from the use of a visual language. The investigation of the hypothesis that the prototype completed can prove that the concept of STI BLOP, once developed, will provide an interactive and rich learning environment for students in their learning process.

Key words: Programming Logic. Algorithms. Intelligent Tutoring System. Case Based Reasoning.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS

API – *Application Programming Interface*

APPS – *Application*

BLOP - *Bloco de Lógica de Programação*

CAI – Instrução Auxiliada por Computador

CAL – Aprendizagem Auxiliada por Computador

CBI – Instrução Baseada em Computador

IA – Inteligência artificial

IDE – *Integrated Development Environment*

MBR – Raciocínio Baseado em Modelos

MOPs – *Memory Organization Packets*

RBC – Raciocínio Baseado em Casos

STI – Sistema Tutores Inteligentes

ZDP – Zona de Desenvolvimento Próxima

SUMÁRIO

1 INTRODUÇÃO

A apropriação do conhecimento referente à programação de computadores é parte essencial do curso de computação. O ensino dessa disciplina também é incorporado a muitos outros cursos como Engenharia de Produção e Engenharia Civil, por exemplo, devido a sua importância para essa área das ciências exatas. No entanto, muitos alunos iniciantes nos cursos relacionados às áreas das ciências exatas e com a computação encontram nessa disciplina um obstáculo de difícil superação. Segundo Santos (2008), nos cursos de computação isso é evidenciado no alto índice de estudantes que abandonam o curso, na diminuição da autoestima e geração de apatia em grande parte dos estudantes, motivadas pela dificuldade de assimilação do conteúdo acarretando, assim, uma alta taxa de reprovação. Por tanto, é necessário encontrar meio de tornar esse assunto menos traumático e mais interessante para os alunos.

Atualmente, temos que considerar que é grande e diverso o número de estudantes que se mostram interessados pelos cursos relacionados a área de computação. No entanto, muitos dos que ingressam devido a distorção no processo de escolarização apresentam dificuldade para resolver problemas lógicos, ou seja, na disciplina de programação. Diante dessa realidade, a princípio, decidiu a pensar uma metodologia na perspectiva de uma tutoria individualizada para cada aluno que a priori parece ser um meio adequado para abordar este problema, porém não é uma alternativa viável financeiramente. Mediante essa consideração uma solução muito mais viável seria usar Sistemas Tutores Inteligentes, que supririam a necessidade de se oferecer um ensino personalizado. Para Botelho (2008), os STI se apresentam como os principais ambientes para auxiliar o processo de aprendizagem de programação. Tendo em vista que STI’s são, em muitos aspectos, semelhantes aos tutores humanos.

Há que ponderar que a programação tem sido um domínio muito produtivo para a construção e evolução da maioria dos aspectos do campo de STI, incluindo o desenvolvimento de princípios sobre a modelagem do módulo do estudante, que representa o conhecimento e as habilidades dos alunos em um dado momento. A modelagem do módulo de domínio que é a representação do conhecimento sobre o domínio que se deseja ensinar, e bem como a aplicação de princípios pedagógicos

que possuem as estratégias e táticas de ensino, modeladas pelo módulo pedagógico.

Segundo Sykes (2007) a programação eficaz exige uma série de estratégias e de diagnóstico e resolução de problemas. A maneira pela qual um estudante escreve um código fornece uma visão rica sobre os processos de raciocínio do aluno. Como resultado, a programação prevê um domínio interessante para estudar a aprendizagem e processos cognitivos.

Sendo assim, os STI's são sistemas computacionais de ensino, que oferece tutoria um para um, que unem técnicas de Inteligência Artificial - IA - com teorias pedagógicas para tutorar um aluno em um determinado domínio, como por exemplo a programação de computadores, alterando sua interação com o aluno com base em características pessoais e individuais (WOOLF, 2009). Atualmente, os STI's são cada vez mais utilizados para promover a educação permitindo aos alunos ter mais uma opção de educação dinâmica e individualizada, mudando a natureza no processo de ensino e aprendizado. Neste contexto, um STI deve ser uma ferramenta de apoio à aquisição do conhecimento e que seja de caráter estimulante e integrador para o aluno.

Considerando essas preliminares observações sobre a disciplina e o ensino de programação na computação é que sentimos motivados a desenvolver esse trabalho de pesquisa na busca de soluções mais viáveis ao enfrentamento da dificuldade dos alunos no processo de aprendizagem dos conceitos e técnicas básicas para a construção de algoritmos, que é a base dos cursos de computação, e que se constitui num dos principais fatores que levam os cursos de formação em computação a atingir altas taxas de evasão, Santos (2008).

Assim, a utilização de novas tecnologias nesse processo tem buscado aliar recursos computacionais a fim de produzir um meio de ensino onde o aluno possa aprender fazendo uso dos recursos disponíveis no computador. Segundo Hostins e Raabe (2007), são possíveis identificar duas explicações para a popularidade na construção de ferramentas para auxiliar o processo de ensino e aprendizagem de programação:

- A primeira se refere ao alto índice de problemas de aprendizagem entre os aprendizes de algoritmo que faz com que se intensifiquem pesquisas que busquem contribuir para a construção de ferramentas e metodologias que possam auxiliar e reduzir estes problemas;
- E o segundo fato se refere à vários trabalhos voltados para a área da informática na educação, e em especial de Sistemas Tutores Inteligentes, têm algum tipo de ligação direta ou indireta com as disciplinas de programação e buscam domínios bem formalizados para um bom emprego de teorias e técnicas inovadoras.

Sendo assim, sistema que ensina programação precisa fornecer exercícios práticos para os alunos. Para que aprendam a partir do sistema, faz-se necessário que recebam o *feedback* em suas soluções dos exercícios. Por outro lado, temos que considerar nesse contexto um desafio importante, pois, um problema de programação raramente tem uma solução única. Para que um sistema seja eficaz, é necessário que seja capaz de lidar com certas variedades de soluções alternativas para um determinado exercício de programação. No propósito de atingir este objetivo utilizaremos uma técnica de Inteligência Artificial.

Na literatura contemporânea sobre IA deparamos com diferentes técnicas de representação de conhecimento utilizadas para implementação da modelagem do módulo de domínio de um STI, dentre tais podemos destacar a Lógica Nebulosa, Redes Bayesianas, Redes Neurais, Lógica Fuzzy e também o Raciocínio Baseado em Casos – RBC (termo original, *Case-based Reasoning* - CBR). O princípio básico dos sistemas de raciocínio baseado em casos é buscar armazenar de forma organizada problemas com suas soluções de tal forma que possam ser utilizados para novas resoluções adaptando soluções de problemas anteriores já resolvidos.

Wangenheim, Wangenheim e Rateke (2013), definem RBC como sendo uma abordagem para a solução de problemas e para o aprendizado baseado em casos e experiência passada. Os sistemas RBC têm como princípios fundamentais o armazenamento e organização de problemas com suas soluções, em bases de casos, e sua utilização para resolver novos problemas ao recuperar e adaptar experiências passadas - chamadas casos. Um novo problema é resolvido com base na adaptação de soluções de problemas similares já conhecidas.

Esses sistemas RBC convertem, também, as soluções do aluno em um conjunto de predicados que são, então, comparados contra um conjunto de casos armazenados em uma base de casos, que também é descrito como um conjunto de casos. Os casos usados para identificar similaridades com a resolução de problemas anteriores nos exercícios de programação que foram cumpridos corretamente e, assim, fornecer *feedback* relevante para o aluno.

Um STI contém vários módulos, dentro os quais podemos destacar o módulo do estudante, que tem a finalidade de personalizar a interação do sistema para com o aluno. Para dar conta disso, o sistema precisa ter algumas informações preliminares sobre o aluno como: sua idade, sexo, capacidades, emoções e outras características. Porém, o objetivo deste trabalho não está focado no módulo do estudante. No que tange ao universo desta pesquisa priorizaremos as características dos alunos que estão mais relacionadas com o seu nível atual de conhecimento no processo de desenvolvimento e construção de algoritmos, no sentido de procurar entender como os alunos pensam regras e alternativas para resolver problemas algorítmicos específicos.

Ponderemos que muitos STI's fazem uso de *feedback*, que são funções importantes dentro desses sistemas, durante o processo de aprendizagem, porém, para fornecer um *feedback* ao aluno, requer que o sistema tenha uma noção do quanto ele irá contribuir para a evolução do aprendizado desse aluno. Neste STI o *feedback* toma grande importância uma vez que ao usá-lo possibilita ajudas na resolução de problemas que se encontram estruturados em vários níveis de atividades dentro do sistema. Entendemos, porém, que se o *feedback* fosse personalizado com o nível de conhecimento atual de cada aluno e com suas características poderá proporcionar um maior aprendizado. No entanto, nossa pesquisa não se propõe concentrar em um módulo pedagógico avançado, mas sim em um módulo de domínio que contribua para resolução efetiva de problema algorítmicos, assim não será fornecido tal personalização.

O STI BLOP, também, fornece instruções e orientações aos alunos no que tange ao próximo e melhor exercício que ele possa tentar resolver, conforme o nível de atividade do sistema no qual o aluno se encontrar. Isto é feito dividindo a matéria do assunto em níveis e armazenando uma estimativa probabilística do conhecimento

de cada aluno referente a cada nível. As estimativas são atualizadas com base em cada solução que o estudante envia para os exercícios.

O nível de conhecimento de cada tema e os temas abrangidos por cada exercício é utilizado para encontrar o exercício que tem o menor número de dificuldades que não são conhecidos pelo aluno. Isso garante que o aluno vá aprender algo novo por tentar este exercício, reduzindo a quantidade de novos conceitos a fim de não sobrecarregar o aluno.

Portanto, o objetivo deste trabalho de pesquisa é propor o STI BLOP para auxiliar ao processo de ensino e aprendizagem de alunos no domínio de lógica de programação, com a finalidade de facilitar o desenvolvimento de técnicas que cooperam para a construção de soluções logicamente válidas e coerentes, com capacidade de resolução com qualidade de problemas que se desejam programar e, assim, contribua para identificação de dúvidas e erros dos alunos na construção de algoritmos.

1.1 Objetivos

1.1.1 Objetivo Geral

Propor um Modelo de Sistema Tutor Inteligente para o ensino no domínio de lógica de programação e comprovar, através do desenvolvimento de um protótipo, que a utilização de um STI para auxiliar o processo de ensino e facilitar o desenvolvimento de técnicas que cooperam à produção de soluções logicamente válidas e coerentes, contribuindo, assim, à identificação de dúvidas e erros dos alunos na construção de algoritmos.

1.1.2 Objetivos Específicos

1. Fazer uma breve revisão sobre Sistemas Tutores Inteligentes (STI);
2. Identificar uma técnica de representação do conhecimento e raciocínio – Raciocínio Baseado em Casos – RBC –, para ser utilizada nesta pesquisa;
3. Investigar e analisar alguns STI’s como apoio ao ensino e aprendizagem de algoritmos;

4. Definir os componentes básicos necessários à especificação de um STI para o ensino no domínio de lógica de programação;

5. Demonstrar a viabilidade da aplicação na criação de um protótipo no domínio de lógica de programação, demonstrando que o estudante que utiliza o STI BLOP, como auxílio no desenvolvimento de problemas de lógica de programação, pode errar menos na solução destes problemas;

6. Demonstrar que através da aplicação do STI BLOP, o professor dispõe de uma ferramenta computacional com estatísticas de aprendizado da disciplina de lógica de programação, em resoluções dos problemas realizados pelos alunos.

1.2 Justificativa da Pesquisa

A internet, na contemporaneidade, se tornou popular em todo o mundo, e seu uso tem despertado em uma grande quantidade de pessoas interesse por essa ferramenta, por outro lado, as dificuldades na apropriação dos conhecimentos referentes a programação de computadores têm sido recorrentes a muitos que transita por esse universo tanto como usuário ou como estudante em busca de uma profissionalização. No contexto da formação é extremamente significativo o número de linguagem de programação que podem ser usadas no processo de ensino e aprendizagem da disciplina de algoritmos, essa realidade se reflete no aprendizado de programação de computadores, e coloca os aprendizes na busca de uma boa base de conhecimento de conceitos referentes ao domínio de lógica de programação.

Sendo assim, a apropriação de conhecimento no domínio de lógica de programação é essencial para quem quer aprender a programar. E a linguagem visual de programação se apresenta como uma ferramenta facilitadora para os neófitos nesse campo de conhecimento, uma vez que, a linguagem visual é uma linguagem de programação popular para ensinar iniciantes a programar. Nessa perspectiva, todo curso projetado para ensinar novatos a programar de forma eficiente, deve abordar o conhecimento referente ao domínio de lógica de programação e particularidades de lógica que vão para além dos conceitos de

programação, cujo o objetivo é garantir que os alunos aprendam de forma eficaz e tenham um bom desempenho no decorrer do curso.

Este estudo que apresentamos tem como objetivo construir uma solução para o problema do ensino e aprendizagem de alunos que estão iniciando nos cursos de computação, no domínio de lógica de programação, no processo de desenvolvimento dos algoritmos, criando, assim, meios que auxilie a pensar em regras e possibilidades de desenvolvimento de algoritmos, de uma forma eficaz e viável.

1.3 Metodologia

Na perspectiva de uma abordagem metodológica esta pesquisa é classificada como pesquisa aplicada, uma vez que tem como objetivo apresentar uma ferramenta para aplicação prática de um problema específico, utilizando o STI BLOP para o ensino no domínio de lógica de programação. Segundo Marconi e Lakatos (2002) “...a pesquisa aplicada, como o próprio nome indica, caracteriza-se por seu interesse prático, isto é, que os resultados sejam aplicados ou utilizados, imediatamente, na solução de problemas que ocorrem na realidade...". O artefato resultante desta pesquisa é um modelo de um protótipo de sistema tutor inteligente chamado STI BLOP para ensinar alunos conhecimentos referentes ao domínio de lógica de programação.

Inicialmente, a pesquisa bibliográfica realizada buscou referências teóricas que buscassem compreender as dificuldades dos alunos na disciplina relacionado a Algoritmo e sobre a evasão de alunos nos cursos de Computação – índices esses, que justificam a necessidade do presente estudo. O trabalho apresenta ainda uma revisão de literatura sobre Sistemas Tutores Inteligente e Raciocínio Baseado em Casos. E serão apresentados e descritos alguns ambientes de apoio ao ensino de algoritmos. Esses ambientes servirão de base para o desenvolvimento de um modelo de um sistema de apoio ao ensino e aprendizagem no domínio de lógica de programação.

A pesquisa circunscreve, também, nas características de estudos de casos uma vez que, para avaliar como a aplicação do STI BLOP pode contribuir com o processo de ensino e aprendizagem, será colocado a disposição, do professor e do

aluno, uma ferramenta computacional com funções de edição de algoritmos e coleta de dados estatísticas de aprendizado dos alunos oriundos da sua interação com o sistema.

1.4 Limitações da Pesquisa

Este trabalho limita-se em desenvolver e aplicar um protótipo de STI utilizado como técnica de IA o Raciocínio Baseado em Casos - RBC, que faz uso de base de casos. Sendo assim o foco principal do trabalho foi o componente Módulo do Domínio do STI, tendo como consequência uma modelagem simplificada do Módulo do Estudante e Módulo Pedagógico. Em relação às limitações desta pesquisa, há que considerar também, que o conteúdo a ser ensinado através do STI BLOP ainda é muito restrito a conceitos iniciais em lógica de programação e, portanto, incompleto, acarretando a ineficiência do STI em auxiliar o aluno de forma ampla. Outro fator é que o retorno dos casos mais similares pode ser prejudicado, ou até mesmo não ser encontrado nenhum caso similar ao problema a ser resolvido pelo aluno dificultado o *feedback* e, consequentemente, deixando de contribuir para o aprendizado efetivo do mesmo.

1.5 Estrutura da Dissertação

A dissertação está organizada em 5 capítulos. O primeiro capítulo apresenta a introdução, objetivos, justificativas da pesquisa, metodologia e delimitações do estudo. No segundo capítulo concentramos na problematização do trabalho, no qual apresentamos uma introdução sobre algoritmos e lógica de programação, a dificuldade na apropriação do conhecimento referente a algoritmos, algumas formas de representação e as estruturas básicas dos algoritmos bem como sobre linguagem de programação visual.

No terceiro capítulo apresentamos alguns conceitos e paradigmas do uso de Sistemas Tutores Inteligentes no ensino de algoritmos, os tipos STI’s para o ensino no domínio de lógica de programação, os problemas no aprendizado de algoritmo e os trabalhos já propostos. Já o quarto capítulo versamos sobre alguns conceitos, definições, características, componentes, histórico da evolução dos sistemas RBC e ainda as principais formas de representação do conhecimento de Raciocínio Baseado em Casos.

O quinto capítulo tratamos da especificação e o desenvolvimento do protótipo deste trabalho e finalizamos apresentando as considerações finais e proposta para trabalhos futuros.

2 ENSINO E APRENDIZAGEM DE ALGORITMOS

Considerando que este estudo versa sobre o Modelo de um Sistema Tutor Inteligente, nesta parte do trabalho apresenta temas que darão fundamento teórico para o desenvolvimento do modelo computacional do STI BLOP que vai ao encontro do ensino de algoritmos e lógica de programação voltado para a área de informática. Transitamos pelos conceitos da disciplina de algoritmos, pela dificuldade no ensino e aprendizagem de programação bem como pelas técnicas utilizadas para o processo de ensino de lógica de programação, que são temas amplamente discutidos em referência nacional e internacional. Será apresentado ainda neste capítulo, a API (*Application Programming Interface*) *Blockly*, que será usada para o desenvolvimento da linguagem visual de código a ser utilizada nesse trabalho de pesquisa.

2.1 ALGORITMO: Visão geral

Algoritmo é a descrição de um conjunto de comandos que, uma vez obedecidos, resultam numa sucessão finita de ações. Enquanto área disciplinar, é uma das mais importantes dos cursos relacionado com a área de computação, pois a partir dos seus conhecimentos apropriados pelos alunos, o futuro acadêmico do aluno estará ou não garantido, a medida em que as disciplinas relacionadas com programação estão presentes durante todo o curso.

De acordo com Forbellone (2005), quando estamos elaborando um algoritmo devemos especificar ações claras e precisas, que a partir de um estado inicial, após um período de tempo finito, produzem um estado final previsível e bem definido. Isto significa que o algoritmo fixa um padrão de comportamento a ser seguida, uma norma de execução a ser trilhada, com vistas a alcançar, como resultado final, a solução de um problema, garantindo que sempre que executado, sob as mesmas condições, produza o mesmo resultado. Porém, segundo Iepsen (2013), são vários os caminhos que podem levar a uma solução satisfatória para a resolução de um problema algorítmico, sendo que um problema pode ser resolvido a partir de diferentes algoritmos e todos igualmente corretos, ou seja, um problema pode apresentar diferentes versões de solução.

Um algoritmo tem por objetivo representar mais fielmente possível o raciocínio envolvido na lógica de programação e, dessa forma, permite-nos abstrair de uma série de detalhes computacionais, que podem ser acrescentados mais tarde. Assim, podemos focalizar nossa atenção naquilo que é considerando importante - a lógica da construção de algoritmos. Outra importância da construção dos algoritmos é que, uma vez concebida uma solução algorítmica para um problema, esta pode ser traduzida para uma das muitas linguagens de programação existentes hoje.

Segundo Engelbert (2013, *apud* IEPSEN), com referências no trabalho de Knuth (1968), destaca cinco propriedades que são amplamente aceitas como requisitos para a construção de algoritmos:

- ✓ Finitude - um algoritmo deve sempre terminar após um número finito de etapas.
- ✓ Definição - cada passo de um algoritmo deve ser definido com precisão; as ações a serem executadas deverão ser especificadas rigorosamente e sem ambiguidades para cada caso.
- ✓ Entrada - valores que são dados ao algoritmo antes que ele inicie. Estas entradas são tomadas a partir de conjuntos de objetos especificados.
- ✓ Saída - a exibição dos valores resultantes das ações do algoritmo relacionadas com as entradas especificadas.
- ✓ Eficácia - todas as operações a serem realizadas no algoritmo devem ser suficientemente básicas que podem, em princípio, ser feitas com precisão e em um período de tempo finito por um homem usando papel e lápis.

Como bem já referimos, no mundo atual existem várias linguagens de programação e cada uma possui uma sintaxe e um conjunto de comandos, com os quais podemos escrever programas para uma infinidade de propósitos, tais como automação comercial, gestão de negócios, *websites*, etc. pois, para Iespsen (2013), o domínio de tais linguagens são essenciais e fundamentais para que os alunos possam desenvolver sistemas para inúmeras atividades, porém se apresentam como um obstáculo para o aprendizado de iniciantes na área de computação.

Vejamos algumas dessas linguagens e as etapas desses processos de programação:

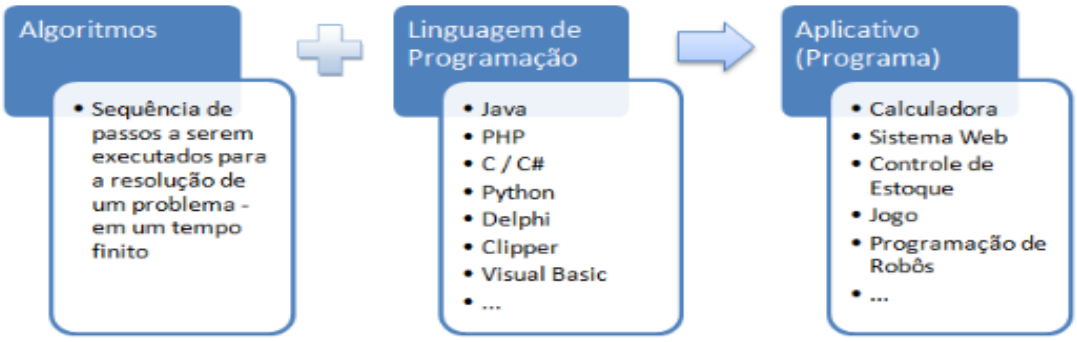


Figura 1 - Apresentação de linguagens e processo de construção de programas (IEPSEN, 2013).

Entretanto ao montar um algoritmo, precisamos primeiro dividir o problema representado em três fases fundamentais. Onde temos: a ENTRADA: que são os dados de entrada do algoritmo; o PROCESSAMENTO que são os procedimentos utilizados para chegar ao resultado final e a SAÍDA que são os dados já processados.

Quanto a essa questão, na Figura 2, apresentada a seguir, podemos visualizar uma analogia básica com o processo de **entrada**, **processamento** e **saída** de informações humanas.

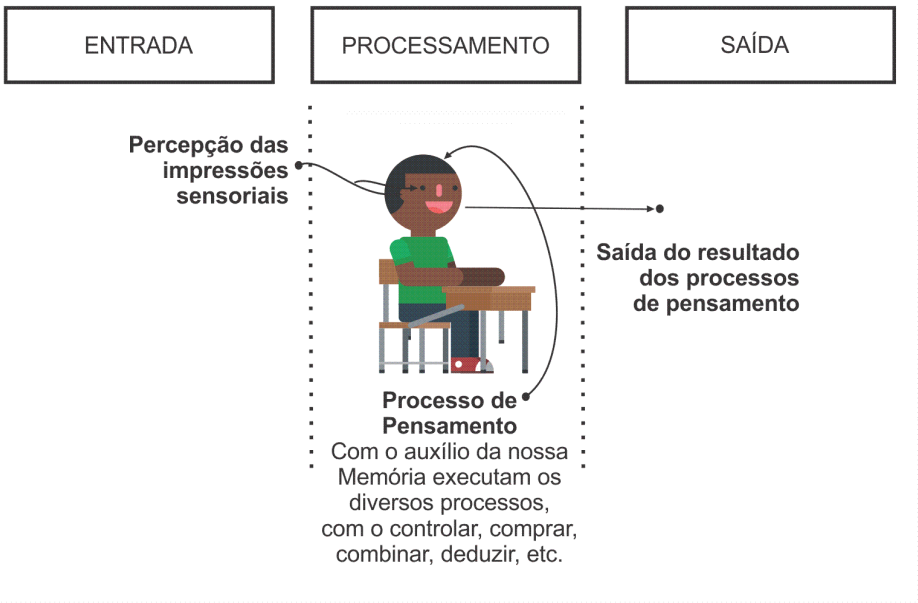


Figura 2 - processo de entrada, processamento e saída de informações humanas.

Pois, para Iepsen (2013), o ensino de algoritmos, embora geralmente sejam adotadas práticas de construção de programas com níveis menores de complexidade, existem cinco primeiras etapas que devem ser realizadas pelo aluno. Sendo elas:

a). Entender o problema;

b). Planejar a lógica;

c). Codificar o programa;

d). Usar o *software* para traduzir o programa para linguagens de máquina e;

e). Executar e testar o programa.

Na execução do programa, a partir dos dados de entrada e saída, podemos verificar se o algoritmo construído pelo aluno está correto ou não. Porém, para contribuir a um ensino efetivo, este trabalho de pesquisa, tem como o foco e preocupação de entender como se dá o processo de desenvolvimento do algoritmo pelo aluno. Para isso, o sistema procurou chegar à conclusão de que aluno sabe desenvolver o algoritmo mas não está resolvendo o problema de forma logicamente

válida, ou seja, ele entende todo o processo de desenvolvimento do algoritmo. E para isso, acreditamos que o *feedback* se apresenta como um importante instrumento para guiar o aluno neste processo. Pois, dois são os tipos de erros que podem ocorrer nos Algoritmos:

a) erros de sintaxe – como declaração incorreta de tipos e nomes de variáveis, sintaxe de comandos ou fluxo da programação;

b) erros de lógica – quando o programa é executado, mas não apresenta os resultados esperados.

No primeiro tipo, o ambiente informa ao aluno o erro, enquanto que no segundo, é o aluno que deve perceber que o seu algoritmo está incorreto. Porém, com a linguagem visual de código acreditamos que se torna quase que impossível o aluno cometer erros de sintaxe[1], tendo vista não será necessário o uso de sintaxe de comandos pois estão todas pré-definidas no ambiente.

2.2 Lógica de Programação

Na conceituação de lógica de programação Forbellone (2005) diz que:

> Significa o uso das leis do pensamento, da “ordem da razão” e de processos de raciocínio e simbolizações formais na programação de computadores, objetivando a racionalidade e o desenvolvimento de técnicas que cooperam para a produção de soluções logicamente válidas e coerentes, que resolva com qualidade os problemas que se deseja programar (FORBELLONE, 2005 p. 02,).

Ainda de acordo com Forbellone (2005) a lógica é a arte de bem pensar, que é a ciência das formas do pensamento. Visto que a forma mais complexa do pensamento é o raciocínio, a lógica estuda a correção do raciocínio. Podemos ainda dizer que a lógica tem em vista a ordem da razão, isto dá a entender que a nossa razão pode funcionar desordenadamente. Por isso a lógica estuda e ensina a colocar ordem no pensamento.

[1] O erro de sintaxe refere-se a comandos digitados de forma incorreta, falta de algum detalhe próprio da linguagem que está sendo usada.

A lógica facilita o raciocínio na construção e entendimento do algoritmo, mostrando que ele está muito mais presente em nosso cotidiano do que imaginamos. Na computação o algoritmo é essencial. E algoritmos nada mais é que uma sequência lógica, finita e definida de instruções que mostra os procedimentos necessários para a resolução de um determinado problema.

Diante do exposto, a lógica é fundamental na construção de algoritmos, pois tem, por objeto de estudo, as leis gerais do pensamento, e as formas de aplicar essas leis, corretamente, na resolução de uma tarefa. Na computação, ela é usada para representar problemas e obter soluções.

A lógica de programação e a construção de algoritmos são conhecimentos fundamentais e essenciais para os alunos que frequentam os cursos relacionados a área da computação. Pois para resolver um problema no computador é necessário que seja, primeiramente, encontrada uma maneira de descrever este problema de uma forma clara e precisa.

Para escreve estes problemas, usamos comandos que são necessários para montagem de um programa para computador, e são escritos seguindo um padrão definido por uma linguagem de programação. Uma linguagem de programação é como um idioma usado para escrever programas para computador.

Assim, segundo Mortari (2001) a lógica é a ciência que estuda princípios e métodos de inferência, tendo o objetivo principal de determinar em que condições certas coisas se seguem (são consequência), ou não, de outras.

2.3 Dificuldade no ensino e aprendizagem de lógica de programação

Nos cursos da área de computação, como bem já referimos, as disciplinas relacionadas com o ensino e aprendizagem de programação de computadores são essenciais para todas as carreiras ligadas a área de informática. Essas disciplinas ocupam papel de destaque e são essenciais para o bom desempenho dos alunos no decorrer do curso uma vez que são importantes para a formação daqueles que terão no desenvolvimento de *software* o produto final do seu trabalho. Sendo assim, a aprendizagem dos conceitos e de programação em si ocorrem, praticamente, no decorrer de todo o curso, pois, o baixo índice de assimilação dos estudantes nessas

disciplinas cujos requisitos exigem o conhecimento de programação tem sido um grande problema enfrentado em muitas instituições de ensino no Brasil (PIMENTEL, 2003).

Podemos observar que nos cursos de computação as disciplinas que trabalham com conhecimentos referentes a programação, uma certa falta de interesse de parte dos alunos afetando assim o seu aprendizado. E isso se deve ao fato dessa disciplina ter como ponto fundamental uma grande carga de conceitos abstratos que perpassa em todo o processo de ensino e aprendizagem dessas disciplinas relacionadas a área de programação de computadores, onde a sofisticação dos ambientes de programação, principalmente das linguagens de programação, e da máquina em si, tendem a dificultar a elaboração do pensamento lógico dos alunos, devido a complexidade desses conhecimentos. Assim, para minimizar a falta de interesse dos alunos uma boa alternativa seria colocar aos iniciantes nessa disciplina, ambientes com uma boa carga de simplicidade (ALMEIDA *et al...*, 2005).

Apesar de várias metodologias propostas e disponíveis terem verificado melhorias nos índices de aprendizado no domínio de algoritmo, prevalece ainda, em grande parte das metodologias atuais, déficit que dificulta tratar cada aluno de maneira diferenciada. Ou seja, as metodologias geralmente são aplicadas de maneira uniforme às turmas inteiras.

Mendes (2002), destaca alguns fatores que contribuem para a dificuldade no processo de ensino e aprendizagem em disciplinas que envolvem conceitos referentes a programação, dentre elas elencam:

- Algoritmos e programas computacionais são processos dinâmicos, isso implica que os métodos tradicionais de ensino dessa disciplina não são mais adequados tendo em vista às necessidades de se aprender conceitos dinâmicos, e não apenas fazendo uso de técnicas estáticas.
- Outra fonte de dificuldade seria a natureza abstrata das disciplinas de programação, tendo em vista que muitos dos alunos não conseguem abstrair a estruturação de um programa e nem consegue aprender como funciona para resolver um certo problema.

- Outro ponto é que as disciplinas de programação exigem em sua aprendizagem muito estudo baseado na prática o que a diferencia de outras disciplinas. Assim, o aluno tem que ter consciência que se aprende programação, fundamentalmente, resolvendo exercício e não somente observando como se faz, pois isso exige treino, persistência e um bom nível de conhecimento e práticas de resolução de problemas.

Nessa perspectiva, esses fatores geram, frequentemente, desorientação e desinteresses por parte de muitos dos alunos, exigindo um acompanhamento e orientação que dificilmente o professor terá para disponibilizar e que os métodos tradicionais nem sempre dão conta de suprir (MENDES, 2002).

Segundo Rodrigues Junior (2004), estas disciplinas possuem um elevado índice de reprovação e desistência nas instituições de educação do Brasil, devido a esse processo de apropriação do conhecimento referente a essas disciplinas. Devido sua complexidade, tem sido foco de professores, preocupados com a melhoria no processo de ensino e aprendizagem, sugerindo assim uma modificação tanto na parte didática quanto metodológica de sua apresentação.

Outro ponto, há considerar, é que os alunos, muitas vezes, já chegam desmotivados em sala imbuídos do preconceito de que a disciplina será um obstáculo difícil de à ser superado, além de alguns professores reforçar essas pré-noções, quando chega a afirmar que o aprendizado é complicado e que as avaliações aplicadas serão muito difíceis (RODRIGUES JUNIOR, 2004). Muitos estudantes vêm para o seu primeiro curso de programação com a ideia pré-concebida de que a programação é um assunto difícil (GOMES, 2007). Outros não conseguem entender a importância do aspecto prático de programação de computadores e tenta passar o assunto, simplesmente memorizando livros didáticos, sem exercitar o código (GOMES, 2007). Tal orientação inadequada, com base em práticas e atitudes incorretas, torna a programação um assunto difícil para alguns alunos.

Para Siebra (2009), as disciplinas de lógica de programação, por ser umas das primeiras a serem oferecidas nos cursos de computação, geralmente são formadas por alunos que vem de uma realidade de formação básica marcada de

lacunas na sua formação inicial e isso impacta o desempenho do discente no curso de programação. No decorrer da disciplina, nem sempre esse aluno poderá contar com um acompanhamento e atendimento individualizado por parte do professor.

Tendo em vista essa diversidade de nível de formação inicial dos alunos, justifica pensar ferramentas com possibilidade de enfrentar essa problemática que evite os alunos a serem submetidos as mesmas condições de ensino, e assim evitar a apresentarem resultados distintos como bem chama atenção Pimentel (2003), reforça a necessidade do uso de técnicas variadas que permitam ampliar positivamente os resultados de ensino.

Como uma das formas de facilitar o entendimento dos métodos de construção de algoritmos no início do processo de aprendizagem, o professor deve escolher preferencialmente problemas inseridos no contexto da realidade cotidiana dos alunos (FORBELLONE, 2005). Assim, o professor poderá propor estudos de casos buscando aumentar a complexidade dos algoritmos de forma gradual e restrita à lógica, sem se preocupar com as linguagens de programação, por exemplo.

Pimentel (2003), afirma que é comum ouvir que, “algoritmo e programação não é para todos”. Esta é uma visão simplista daqueles que não questionam a maneira de ensinar. Porém, para uma razoável parcela da comunidade científica, surgem questionamentos sobre como ensinar algoritmo. São pesquisadores preocupados em entender o processo de aprender a programar, detectando falhas e dificuldades deste aprendizado e sugerindo alternativas, de modo a facilitar o aprendizado do aluno nessa disciplina.

2.4 Técnicas utilizadas para o ensino de lógica de programação

Por definição, um algoritmo é qualquer procedimento computacional ou uma sequência de passos bem definida em uma ordem específica transformando uma entrada fornecida em um valor ou conjunto de valores como saída (CORMEN, 2002). E segundo Forbellone (2005), lógica de programação significa o uso das leis do pensamento, da “ordem da razão” e de processos de raciocínio e simbolizações formais na programação de computadores, objetivando a racionalidade e o desenvolvimento de técnicas que cooperam para a produção de soluções

logicamente válidas e coerentes, que resolva com qualidade os problemas que se deseja programar.

Para escrever programas de computador corretos, os alunos precisam entender conceitos abstratos, em seguida, convertê-los em soluções concretas (GOMES, 2007). O problema deve primeiro ser resolvido através de uma abordagem conceitual antes de um programa de computador poder ser escrito usando uma linguagem de programação particular. Ao fazê-lo, os alunos precisam utilizar as habilidades na concepção de programas e pensamento criativo. Pois, ao criar uma solução, eles precisam concentrar-se simultaneamente sobre a sintaxe e a construção do algoritmo (GOMES, 2007). Isto significa que todo o processo requer a interação de muitas capacidades cognitivas, o que torna o processo muito difícil para principiantes.

Porém, existem diversas formas de representação de algoritmo e na construção de algoritmos a apropriação dos conceitos referentes à lógica de programação é essencial, pois para quem deseja trabalhar com desenvolvimento de programas para computadores têm que se apropriar de tais conhecimentos. Lógica de programação pode ser definida como um conjunto de técnicas para encadear pensamentos a fim de atingir determinado objetivo. Para o desenvolvimento de algoritmos temos que usar essencialmente a lógica de programação, e na tentativa de facilitar a apropriação desses conhecimentos, fazem-se usos de algumas técnicas, tais como: o uso de fluxograma, o uso da linguagem de programação de auto nível ou pseudocódigo códigos ou linguagens de programação visual (blocos). Para que se possa desenvolver um algoritmo funcional o aluno deve seguir etapas lógicas fazendo uso de uma das técnicas citadas acima. A seguir apresentamos um exemplo do uso de fluxograma, de linguagem de alto nível e de linguagem visual em bloco. Esse exemplo a seguir, requer que os alunos façam um cálculo do fatorial de um número **N inteiro** que é obtido através da multiplicação de N pelos seus antecessores até se chegar ao número 1.

Para demonstrar uma das formas de solução do algoritmo utilizaremos o fluxograma, que é usado para mostrar de forma gráfica a lógica para resolver problemas, sendo que nesse processo destaca-se passos individuais e o fluxo de execução (VALENTIN, 2009). Assim, o fluxograma é nada mais do que uma

representação gráfica do algoritmo, através de formas geométricas, facilitando a compreensão da lógica utilizada pelo aluno, como bem ilustra a Figura 5, destacando os passos usados no desenvolvimento da lógica de programação na resolução do problema para encontrar o fatorial de 5.

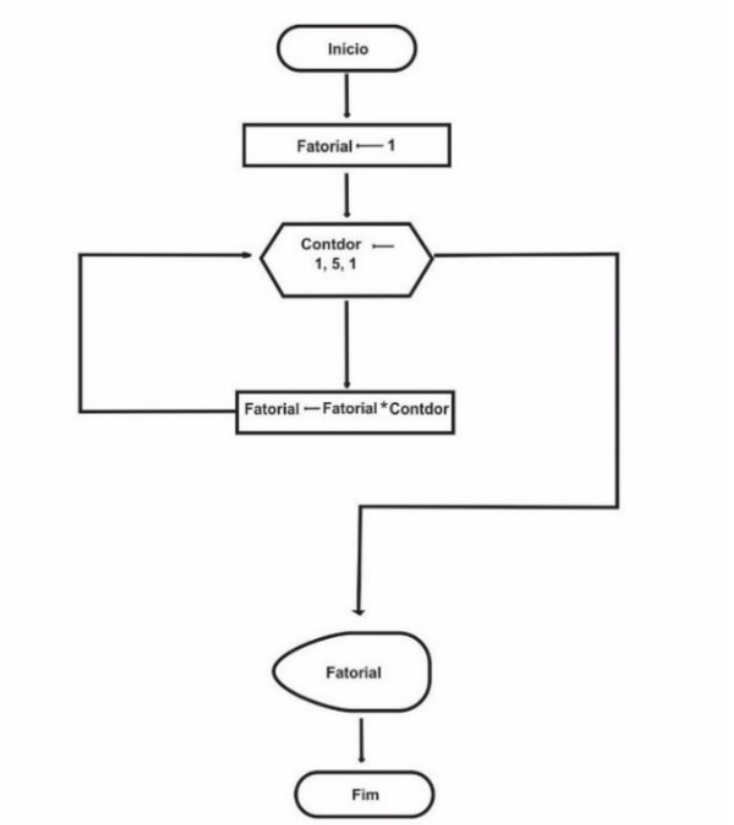


Figura 3 - Fluxograma: cálculo do fatorial de N.

Porém, temos sempre que considerar que computadores não são programados usando desenhos, pois existem, no entanto, algumas linguagens gráficas, que, infelizmente, são raras e pouco utilizadas (VALENTIN, 2009).

As linguagens de programação de auto nível, por outro lado, fazem uso de uma infinidade de palavras adicionais, como “*program*”, “*var*”, “*begin*” e “*end*”, e de um grande número de caracteres, como aspas, vírgula, ponto e vírgula, setas e etc; e que podem, muitas vezes, acarretar um problema a mais para os alunos no aprendizado da disciplina, como bem podemos visualizar na Figura 6.

```
Program FATORIAL;
var
   I, N, FAT: integer;
Begin
      Writeln('Digite um número:');
      readln (N);
      FAT := N;
      For i:=1 to  (N-1)  do
          FAT := FAT * I;
      writeln('O fatorial de  ', N, ' equivale a:  ', FAT);
end;
```

Figura 4 - Linguagem Pascal fatorial de N.

Todas essas palavras e caracteres são necessários para que o programa seja entendido pelo compilador, que é um conjunto de programas que tem por objetivo traduzir um programa escrito em uma linguagem de alto nível, mais próximas da linguagem humana, como Pascal, Java, C, C++ e etc., para uma linguagem de máquina, ou baixo nível, como o *Assembly*, por exemplo. Esta tradução envolve a análise sintática, a qual tem por objetivo verificar se o programa está escrito dentro das regras da linguagem de programação do compilador (VALENTIN, 2009).

Frente ao exposto, fica claro que todo esse excesso e sobrecarga de sintaxe das linguagens de programação torna-se uma barreira a mais no processo de ensino e aprendizagem na disciplina de programação, pois além de se preocupar com o desenvolvimento do seu raciocínio, o aluno, ainda tem que se preocupar com a assimilação da linguagem. Esse é outro ponto que tem que ser levado em consideração quando se fala sobre o ensino e aprendizagem na disciplina de programação.

A linguagem visual em blocos, que consiste na criação de códigos onde o aluno pode arrastar e soltar blocos para escrever programas, mostra-se mais interessante e atraente para a finalidade do ensino na disciplina de lógica de programação, tendo em vista o pouco ou nenhum conhecimento da maioria dos

iniciantes em programação, pois o aluno não precisará se preocupar com a sintaxe da linguagem quando estiver resolvendo suas atividades. A linguagem visual consiste em que cada bloco corresponde a uma linha de código real em linguagem de código, como: *JavaScript, Python, Dart*, PHP dentre outras, que depois de desenvolvida a lógica, o código pode ser exportado em uma das linguagens citadas a cima, sendo que depois o aluno pode utilizar esse código como base para o desenvolvimento de programas mais complexo. Vejamos a exemplificação na Figura 7 que mostra um código em linguagem visual em bloco.

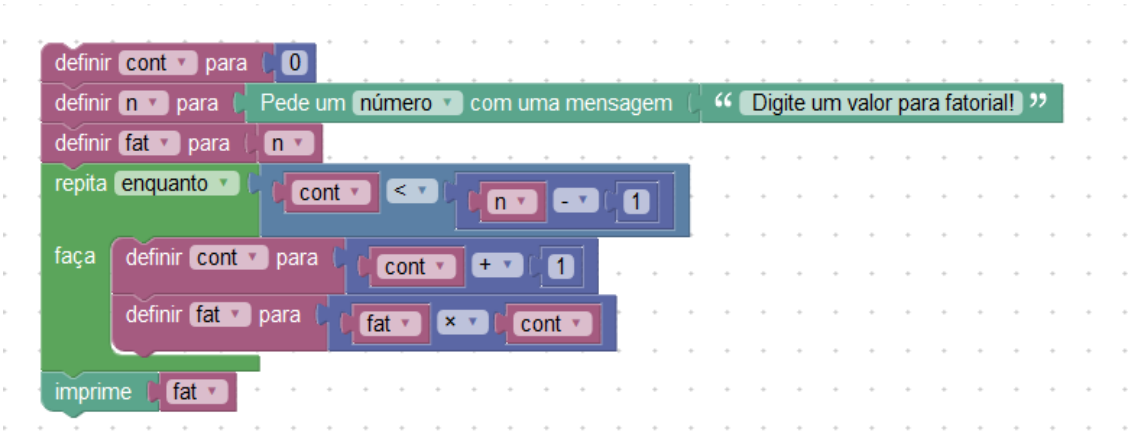


Figura 5 - Linguagem visual em blocos fatorial de N.

Tendo em vista as características da linguagem visual e o foco da disciplina de Lógica de Programação, que está mais interessada na forma conceitual de organização das ideias dos alunos do que no aprendizado da sintaxe de linguagem é que optamos para fazer uso no desenvolvimento do STI BLOP dessa linguagem visual de programação.

Neste sentido, acreditamos que um curso criado para o ensinar de lógica de programação para iniciantes deve abordar a forma de organização de ideias para além dos conceitos de uma linguagem, afim de garantir que os alunos aprendam de forma eficaz o conteúdo a ser transmitido.

Mediante essas condições, este trabalho foi pensando e elaborado com o objetivo de encontrar uma solução para o problema de ensinar lógica de

programação usando uma linguagem visual para uma grande quantidade de indivíduos, de forma eficaz e economicamente viável.

2.5 *API Blockly*

O *Blockly* é uma API para desenvolvimento de editores de programa que faz uso de uma linguagem visual, permitindo aos usuários escrever códigos conectando blocos. Ele é um ambiente baseado na *web* para edição de programação de uma forma gráfica. Os usuários podem arrastar os blocos juntos para construir um aplicativo, assim como se arrasta blocos de montar (lego).

O *Blockly* é uma API disponibilizada pela *Google*, gratuitamente, cujo a finalidade é facilitar a criação e desenvolvimento de apps (*application*), procurando evitar que o desenvolvedor precise decorar comandos, e se preocupar com uma sobrecarga de sintaxe específica de uma linguagem de programação, tais como: aspas, ponto e vírgula, parênteses, chaves e trabalhar com infinitas linhas de códigos. Esta ferramenta está sendo usada largamente para desenvolvimento de aplicações educacionais para auxiliar o processo de ensino e aprendizagem sobre lógica de programação, devido a sua *interface* amigável, intuitiva e lúdica.

O *Blockly* foi adotado por ser uma ferramenta disponibilizada de acordo com a Licença de Atribuição *Creative Commons* 3.0 por se tratar de um *software Open Source* oferece permissões de cópia, modificação e uso, contanto que seja mantida a licença original, os direitos autorais e o fornecimento do código-fonte aos demais usuários.

O *Blockly* está projetado para criar relativamente pequenos *scripts*. No entanto, uma vez que os fundamentos sobre matemática e lógica são apropriados pelos alunos, ele oferece um conjunto rico de blocos de nível superior para desenvolvimento de algoritmos mais complexos. Como já mencionado o *Blockly* é uma API que permite aos usuários escrever programas conectado e juntando blocos. Assim os usuários podem implementar um trecho de programa conforme demonstrado na Figura 3:

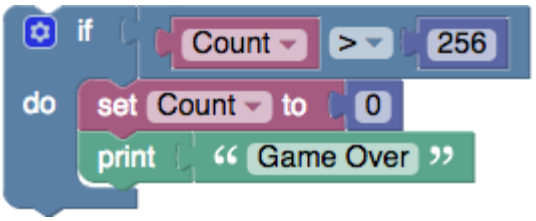


Figura 6 - Trecho de programa em linguagem visual.

Desta feita, o *Blockly* iria gerar o código corresponde em *JavaScript*, *Python*, PHP, *Dart*, ou outra linguagem disponível, possibilitando, assim, que, depois de ser apropriado, tais conceitos, referentes ao uso da lógica de programação, para a criação de programas no qual usuário poderá prosseguir com seu aprendizado fazendo uso de uma outra linguagem. Na Figura 4, mostramos o código gerado pelo *Blockly* referente ao trecho de código da Figura 3.

```
se(Contagem > 256)
   { Contagem  =  0;
     alerta ("Game Over");}
```

Figura 7 - Código gerado pelo *Blockly*.

As características principais do *Blockly* são:

- Executar em um *web-browser*. Não há *downloads* ou *plugins* necessários.
- Código exportável. Os usuários podem extrair seus programas como *JavaScript*, *Dart*, PHP e *Python* ou outra linguagem, para que quando eles superarem o *Blockly* eles possam continuar a aprender em outra linguagem.
- É possível baixar o código em XML e abrir novamente este código parametrizado.
- *Open source*. Tudo sobre *Blockly* está aberta: você pode modificar, baixar, e usar em seus próprios sites.

- Altamente capaz. Com a capacidade de calcular o desvio padrão usando um único bloco, o *Blockly* não é um brinquedo.

Porém, o *Blockly* em si não é uma plataforma educacional, ele é uma API que pode ser utilizada como parte de uma plataforma educacional, ou como parte de um conjunto de negócios, ou como parte de um sistema de jogos, etc.

2.6 Estrutura Básica de Algoritmos com *API Blockly*

O objetivo dessa seção é demostrar a estrutura básica dos algoritmos construídos com a API *Blockly*. Para isso iremos demonstrar a estrutura utilizada na construção de um algoritmo e os comandos básicos de programação com blocos, divididos em estruturas de variáveis, operadores e de estruturas. Os exemplos apresentados estão demonstrados na linguagem visual de blocos.

Para começar a utilizar a ferramenta, o ambiente permite que o aluno navegue através do *menu* de blocos, como bem demonstrado na Figura 8 que representa o *menu* de blocos do STI BLOP.

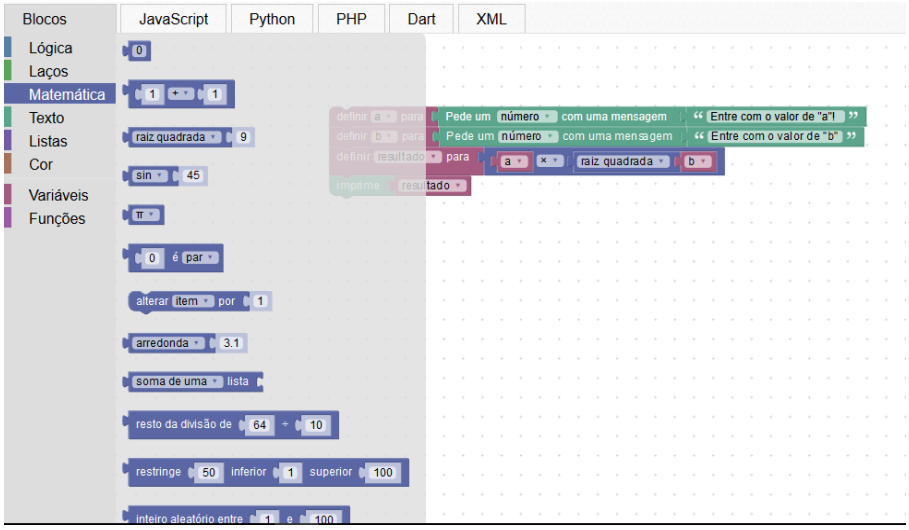


Figura 8 - *Menu* de blocos do STI BLOP.

Para que o ambiente possa compreender os passos necessários para a resolução de um determinado problema, devemos estruturar esses passos de forma

que sejam perfeitamente compreendidos e executados pela ferramenta. Assim, após o aluno montar um código válido ele pede para programa ser analisado e exibir a saída. Tanto a saída dos resultados do código quantos os dados para ser inseridos no processamento do programa serão exibidas e inseridos através de um tela *pop-up* que aparecerá para o aluno, ao ser clicado no botão de compilar programa, na parte superior direta da tela do ambiente.

2.6.1 Variáveis

Variáveis são espações de memória reservados para armazenar informações. Ao iniciar a construção de seu programa e clicar no *menu* na opção variável, será apresentado para o aluno apenas duas opções de blocos “definir...para” e “item”, como visto na imagem abaixo:

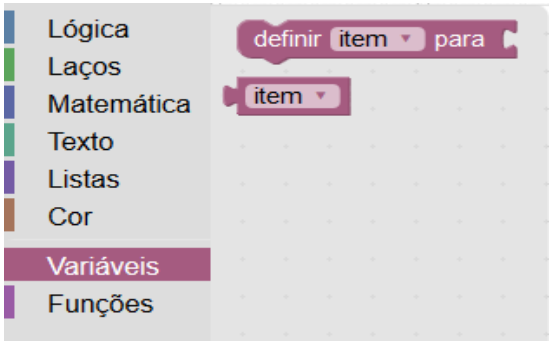


Figura 9 – *Menu* opção variável.

Existem vários tipos de dados e os mesmos podem ser definidos de maneiras distintas nas diversas linguagens de programação existentes. Porém, no ambiente o aluno não precisará se preocupar com a definição dos diversos tipos de variáveis, se inteira, real, caracteres, texto (*string*) ou *booleana*, que são os tipos mais comuns de dados, pois já se encontram todas pré-definidas no *menu* do ambiente. Após iniciar o uso e a criação das variáveis, que serão definidas por tipo de dados associados a um nome que representará a variável, as mesmas serão adicionadas automaticamente no *menu* onde ficaram visíveis no botão variáveis, para que o aluno possa utilizá-la novamente no decorrer do programa.

Aqui, diferente de outras linguagens, como por exemplo C++, é permitido o uso de caracteres especiais tais como acentos e outros caracteres, porém assim como em qualquer outra linguagem não se permite o uso de espaço em branco na formação do nome de uma variável.

2.6.2 Operadores

2.6.2.1 Operadores relacionais

Os operadores relacionais são utilizados para comparar valores, o resultado de uma expressão relacional é um valor *booleano* (Verdadeiro ou Falso). Estes tipos de operadores são usados para realizar comparações entre dois valores de um mesmo tipo. Esses valores podem ser representados por variáveis ou constantes como podemos visualizar na lista que apresentamos os operadores relacionais:

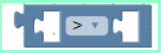
Operadores relacionais	
Descrição	**Símbolo**
Igual a	
Maior que	
Menor que	
Maior ou igual a	
Menor ou igual a	
Diferente de	

Tabela 1 – Lista operadores relacionais.

Nas comparações realizadas utilizando operadores lógicos dá-se o nome de relação. O resultado de uma relação é sempre um valor lógico, ou seja, verdadeiro ou falso.

2.6.2.2 Operadores Lógicos

Os operadores lógicos retornam verdadeiro (V) ou falso (F) conforme a utilização dos operadores. Os operadores lógicos também são conhecidos como conectivos, pois são utilizados para formar novas proposições a partir da junção de outras. Abaixo segue uma lista com os operadores lógicos mais comuns:

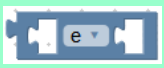
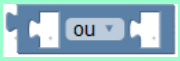
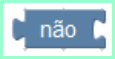
Operadores Lógicos




Tabela 2 – Lista de operadores lógicos.

2.6.2.3 Operadores Aritméticos

Os operadores aritméticos são símbolos que representam operações aritméticas, ou seja, as operações matemáticas básicas. Os operadores aritméticos são utilizados para formar expressões aritméticas. As expressões aritméticas são formadas por operadores aritméticos que agem sobre operandos. Os operandos podem ser variáveis ou constantes do tipo numérico, ou seja, inteiros ou reais. Na tabela 3, a seguir, apresentamos os operadores aritméticos que utilizamos no trabalho que resultou desta dissertação.

Operadores aritméticos	
Descrição	**Símbolo**
Adição	
Subtração	
Multiplicação	
Divisão	
Resto da divisão	

Tabela 3 – Lista de operadores aritméticos.

2.6.3 Estruturas de seleção

Segundo Prus (2001) as estruturas servem para conduzir o fluxo dos dados nos algoritmos, através de testes e condições. Elas diferem umas das outras pela disposição ou manipulação de seus dados ou variáveis. A disposição dos dados em uma estrutura obedece às condições preestabelecidas e caracteriza a estrutura (PRUS, 2001 *apud* SZWARCFITER & MARKEZON, 1994).

Ainda conforme Prus (2001) as estruturas estão divididas em estruturas de decisão e estruturas de controle. As de decisão testam condições que tomam caminhos específicos sem retorno. As de controle ou de repetição permitem executar mais de uma vez um determinado número de comandos.

2.6.3.1 Estrutura Se...FAÇA...SENÂO

A representação dessas estruturas de decisões em nossos programas é feita através do uso dos seguintes blocos de seleção, ou estruturas de decisão. A estrutura de seleção em blocos que utilizamos foi a estrutura SE...FAÇA... SENÃO. A representação dessa estrutura é exibida abaixo:

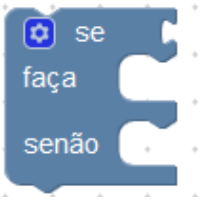


Figura 10 – Estrutura em bloco Se...Faça...Senão.

O funcionamento dessa estrutura se dá da seguinte forma: a palavra reservada “SE” indica o início da estrutura de seleção. Após essa palavra, vem à condição que definirá o bloco a ser executado. Qualquer expressão lógica poderá ser utilizada como condição, pois deverá retornar verdadeiro ou falso. Caso a expressão da condição seja verdadeira, o bloco de instruções FAÇA será executado. Caso contrário, o bloco SENÃO o será. A palavra reservada FIM SE indica o final da estrutura de seleção. Vale Ressaltar que o bloco SENÃO não é obrigatório: caso só queiramos executar algumas instruções se a expressão lógica for verdadeira, podemos omitir o bloco SENÃO.

2.6.3.2 Estrutura REPITA...ENQUANTO...FAÇA

Esta estrutura de repetição permite que enquanto uma determinada condição for verdadeira ou válida os comandos pertencentes à estrutura, são executados. O término da execução dos comandos está vinculado à condição determinada se tornar falsa. A Figura 11 a seguir representa a estrutura do bloco REPITA...ENQUANTO-FAÇA:

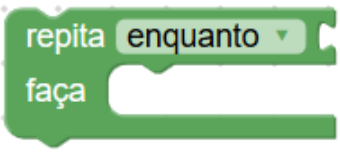


Figura 11 - Estrutura em bloco Repita...Enquanto...Faça.

A palavra reservada ENQUANTO indica o início da estrutura de repetição. Após essa palavra, vem à condição de repetição. Qualquer expressão lógica poderá ser utilizada como condição de repetição, pois deverá retornar verdadeiro ou falso. Caso a condição seja verdadeira, o bloco de instruções será executado. Ao final da execução do bloco de instruções, a condição de repetição é testada novamente e, caso continue verdadeira, todo o bloco será executado novamente e assim sucessivamente até que a condição de repetição se torne falsa.

3 SISTEMAS TUTORES INTELIGENTES

Sistema Tutor Inteligente (STI) é um ambiente computacional de aprendizagem que possuem módulos de conteúdo instrucional que especificam o "que" ensinar e estratégias de ensino que especificam o "como" ensinar (WENGER, 1987).

Os Sistemas Tutores Inteligentes são capazes de acompanhar o aluno durante todo o processo de solução de um problema. Desta forma, enquanto o aluno aprende o conteúdo, o tutor aprende sobre o aluno levantando diversas informações sobre o mesmo, como, por exemplo, facilidades que ele apresenta sobre determinado assunto e dificuldades apresentadas quanto a outros. Com este aprendizado, o tutor pode prover um acompanhamento mais individualizado, apresentando explicações e exercícios mais bem relacionados às habilidades que o aluno apresenta e poderá atuar no auxilio naquilo que não se sente tão seguro ou em área que possui maiores dificuldades.

3.1 História dos Sistemas Tutores Inteligentes (STI)

Os Sistemas Tutores Inteligentes têm sua origem na área de Inteligente Artificial, e em meados dos anos 50 e início dos anos 60 do século XX, muitos pesquisadores acreditavam que os computadores que poderia "pensar" rapidamente tal qual os seres humanos, logo seriam uma realidade. Porém, isso não aconteceu. Alguns acreditavam que com a criação de computadores maiores e mais potentes, eles seriam capazes de realizar qualquer tarefa que estivesse associado com o pensamento humano, como por exemplo, a instrução dos seres humanos (GRAVIDIA e ANDRADE, 2003).

O passo inicial para o surgimento dos primeiros Sistemas Tutores Inteligentes foram os Sistemas de Instrução Auxiliada por Computador - *Computer Aided Instruction* – CAI - ou Aprendizagem Auxiliada por Computador – ou *Computer Aided Learning* – CAL - (GRAVIDIA e ANDRADE, 2003), esses programas eram vistos como um simples programa linear. Essa evolução dos sistemas até os STI's, representamos na Figura 12.

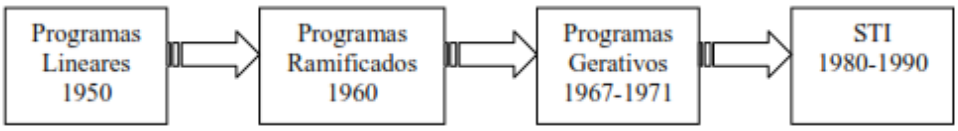


Figura 12 - Evolução dos sistemas de ensino utilizando o computador (GRAVIDIA e ANDRADE, 2003).

Segundo (GRAVIDIA e ANDRADE, 2003), a abordagem de ensino ou os aspectos pedagógicos que tem sido embutido no desenvolvimento de muitos CAI's teve forte influencias das teorias behavoristas, também conhecida como psicologia de estímulos e resposta de B.F. Skinner. Os CAI's caracterizavam-se por mostrar o conhecimento de uma forma linear, isto é, nenhum fator podia mudar a ordem de ensino estabelecida na sua criação pelo programador. A Psicologia Behaviorista defendia que as pessoas funcionavam por estímulos e que para cada estímulo corresponde uma igual resposta. Por esta razão, não se devia permitir que os alunos cometessem erros, já que estes lhe dariam um reforço negativo.

No desenvolvimento de uma sessão de ensino não se levava em consideração, para nenhum fim, o erro do aluno. Acreditava-se que quando uma operação era seguida por um estímulo de reforço, a força da ação era aumentada. Para este fim, a saída do programa de computador dos primeiros CAI's era um frame de texto, que verificava se o conteúdo até aquele ponto havia sido "apreendido". O aluno então dava algum tipo de resposta baseado no que já conhecia ou por tentativa e erro. Finalmente, o programa informava ao aluno se ele estava certo. A ordem pré-definida de tais passos formou o que é conhecido como um *programa linear*. O aluno podia trabalhar usando o material em seu próprio tempo e suas respostas corretas eram recompensadas de imediato.

Nesta perspectiva, os CAI's mostravam-se limitados, pois a apresentação dos assuntos a serem ensinados não se adaptava às necessidades individuais dos alunos, os sistemas possuíam uma habilidade limitada para analisar as intervenções dos alunos bem como de não fornecerem um ensino individualizado (GRAVIDIA & ANDRADE, 2003). Sendo assim, o aluno seguia uma sequência finita e pré-determinada de passos sem com isso estimular o raciocínio frente a diferentes situações.

Uma das necessidades dos sistemas CAI's para que, realmente, possibilitem o aprendizado dos estudantes é possuir flexibilidade no processo de ensino. Isso significa que o sistema precisa ser capaz de deduzir e manter um modelo detalhado sobre o que o aluno sabe e não sabe (WAGNER, 1987).

A posteriori surgiram os programas ramificados ou *programação ramificada* ou ainda programação em árvore, como sucessores dos Sistemas de Instrução Auxiliada por Computador, que era mais adequada para obter um *feedback*, adaptado ao ensino com capacidade de fornecer respostas aos alunos. Estes tinham um número fixo de temas, semelhantes aos programas lineares, mas diferenciavam-se pela capacidade de atuar segundo a resposta do aluno. A melhora oferecida por estes sistemas só foi obtida com a ajuda da técnica de *Pattern-matching*[2] e com a utilização de linguagens de autoria. A técnica de *Pattern-matching* permitiu tratar as respostas do aluno como aceitáveis ou parcialmente aceitáveis, no lugar de totalmente corretas ou incorretas como exigia a proposta de Skinner.

Outra questão a ser considerada foi que os materiais de ensino contidos nos programas lineares eram, em geral, muito extensos e intratáveis por meios clássicos. Por esta razão as "linguagens de autoria" foram desenvolvidas e se caracterizavam por serem linguagens específicas e apropriadas para o desenvolvimento de materiais CAI de forma tratável pelo sistema. Nesta década, de 60 do século XX, a maioria das aplicações educacionais usando computadores adotava o paradigma da instrução programada e seu enfoque centrado no professor, ou seja, o aluno deveria compreender a lição passada pelo professor para posteriormente responder alguma questão relativa ao conteúdo previamente passado (GRAVIDIA e ANDRADE, 2003).

Nos inicios dos anos 1970, surgiram os sistemas gerativos, também chamados de sistemas adaptativos, foram sistemas associados a uma nova filosofia educacional que defendia que os alunos aprendiam melhor enfrentando-se problemas de dificuldade adequada, e não atendendo a explicações sistemáticas, isto é, adaptando o ensino às suas necessidades. Pois há que considerar que os sistemas gerativos são capazes de gerar um problema de acordo com ao nível de conhecimento do aluno bem como construir sua solução e diagnosticar a resposta

[2] A técnica de *Pattern-matching consiste em comparar padrões de strings*

do aluno. Em geral, a solução para um problema concreto não é única, no entanto, os sistemas gerativos criavam só uma solução que era a base de seu diagnóstico (GRAVIDIA e ANDRADE, 2003).

Em respostas às limitações apresentadas pelos sistemas referidos, alguns pesquisadores argumentaram que a solução destas limitações não poderia acontecer sem a utilização de técnicas de IA, nesta perspectiva, foram introduzidas ideias de utilização de técnicas de IA em sistemas CAI's, nascendo assim, ainda nos idos de 1970, os Sistemas Inteligentes de Instrução Auxiliada por Computador - *Intelligent Computer-Aided Instruction* – ICAI ou mais popularmente conhecidos como Sistemas Tutores Inteligentes - *Intelligent Tutoring Systems* – ITS (GRAVIDIA e ANDRADE, 2003). No quadro abaixo, mostramos as principais diferenças entre os sistemas CAI e os sistemas STI.

Aspecto	CAI	STI
Origem	Educação	Ciência da Computação
Bases Teóricas	Skinner (behaviorista)	Psicologia Congnitivista
Estruturação e Funções	Uma única estrutura algoritmicamente predefinida, onde aluno não influi no sequenciamento.	Estruturas subduvidida em módulos, cujo sequenciamento se dá em função das respostas do aluno
Estrutura do Conhecimento	Algorítmica	Heurística
Modelo do Aluno	Avalia a última resposta	Tenta avaliar todas as respostas do aluno durante interação
Modalidade	Tutorial, exercício e prática, simulação e jogos educativos	Socrático, ambiente interativo, diálogo bidirecional e guia.

Tabela 4 - Giraffa (1997, *apud* CARVALHO 2012).

3.2 Arquitetura dos Sistemas Tutores Inteligentes

Segundo Truong (2003) a pesquisa sobre STI são realizadas desde 1960, com o objetivo de fornecer a instrução uma forma eficaz como um tutor humano, mas com baixo custo, sendo que o objetivo do STI é proporcionar os benefícios de uma instrução de um para um automaticamente, preservando o baixo custo. Várias pesquisas mostraram que os STI's são altamente eficazes em aumentar o desempenho e a motivação dos alunos, impactando no desenvolvimento cognitivo, e reduzindo o tempo gasto do aluno para adquirir habilidades e conhecimentos.

Há que considerar que, a arquitetura para um bom funcionamento de um Sistema Tutor Inteligente precisa ter muitos módulos. Uma classificação comum dos módulos que compõem um STI (WOOLF, 2009) como apresentamos na Figura 13. A Figura 13 demonstra como se compreende a funcionalidade de cada um destes módulos, ao considerar uma situação em que o STI forneça um problema para um aluno resolver e quais os passos a seguir. O problema é apresentado ao aluno através do módulo de comunicação que é o que lida com todas as interações entre o aluno e os STI. Em seguida, o aluno entra com sua solução do problema através do módulo de comunicação. O módulo pedagógico, em seguida, considera esta solução em um conjunto de informação obtida a partir dos módulos de estudantes e de domínio. O módulo de domínio contém detalhes sobre o assunto que é ensinado pelo STI e, portanto, contém informações sobre a solução correta para o problema. Com base nessas informações, o módulo pedagógico decide se a solução está correta ou não.

O módulo de estudante contém informações sobre as características do aluno. O módulo pedagógico usa essa informação para decidir que tipo de retorno que deve fornecer ao aluno. Seja qual for a decisão do módulo pedagógico, o *feedback* é fornecido para o aluno através do módulo de comunicação. Enquanto isso, o sistema forma uma base de dados sobre o conhecimento do aluno em determinado assunto que está sendo ensinado pelo problema. Esta informação é atualizada para o módulo do estudante, a fim de ter um modelo mais preciso para o aluno.

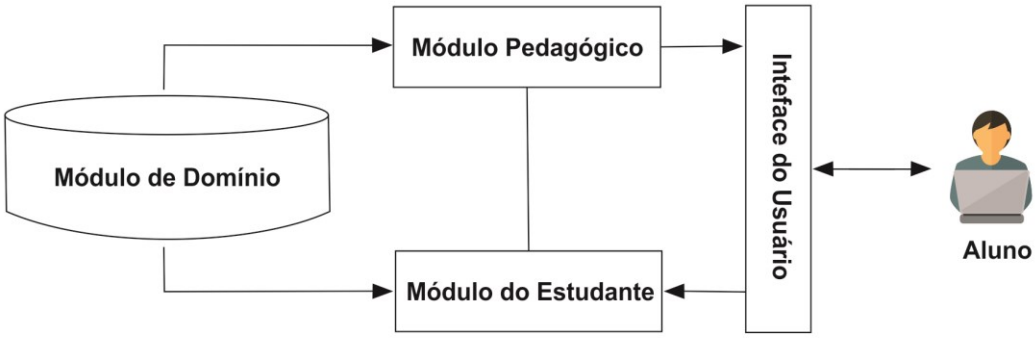


Figura 13 - Principais módulos de um STI (WOOLF, 2009).

Como podemos ver na arquitetura representada na Figura 13, um STI contém um módulo de estudante a fim de personalizar sua interação com o aluno. Para fazer isso, o sistema ideal precisa ter um bom conhecimento sobre o aluno, incluindo sua idade, sexo, capacidades, emoções e outras características. Porém, o foco desta pesquisa não é sobre o projeto detalhado do módulo do estudante. Portanto, o módulo do estudante usado aqui considera apenas as características dos alunos que estão relacionadas diretamente com a aprendizagem, ou seja, o nível atual de conhecimento do aluno no assunto que está sendo ensinado.

Embora esta seja a arquitetura que é usada mais comumente para os STI's, outras arquiteturas foram sugeridas por vários pesquisadores. Destes, a arquitetura sugerida por Pillay (2003) é de interesse, uma vez que foi desenvolvido um STI que ensinam programação. Esta é realmente uma extensão da arquitetura descrita na Figura 13 e contém 10 módulos, como mostrado na Figura 14.

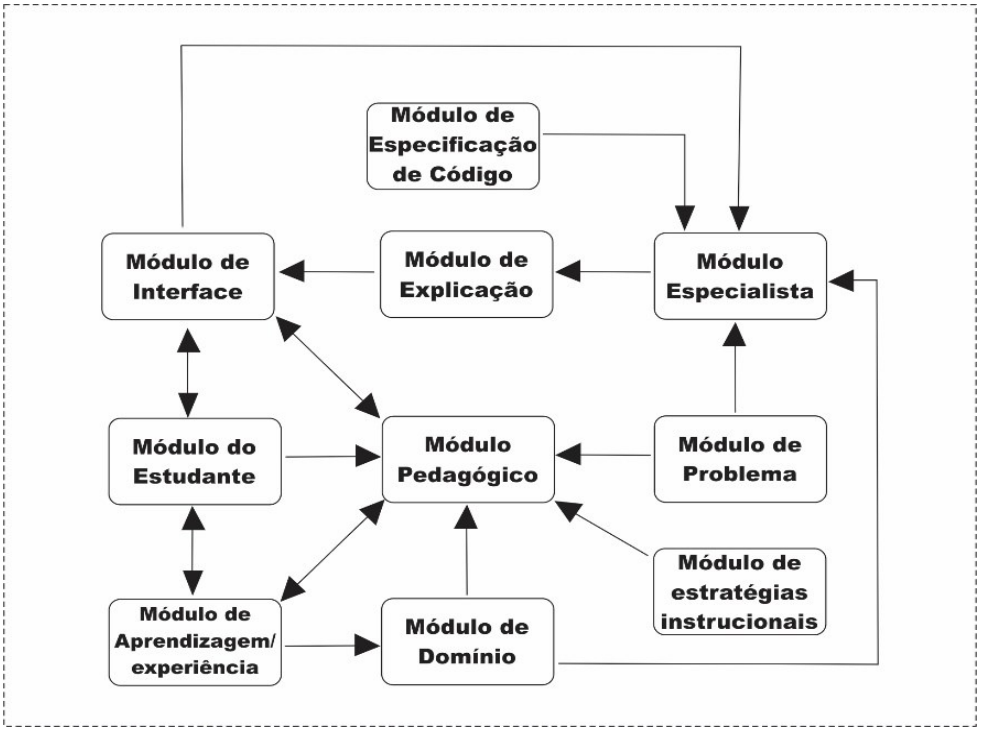


Figura 14 - Arquitetura genérica para um STI de ensino de programação.

O módulo de *interface* nesta arquitetura é o mesmo que o módulo de comunicação na arquitetura anterior. O módulo do domínio na arquitetura da Figura 13 é constituída por módulo do domínio, módulo de problemas, módulo perito e o módulo de especificação de código que aparece na Figura 14. Neste caso, o módulo do domínio é onde as habilidades que são orientadas são armazenadas. O módulo de problemas contém os problemas reais usados para ensinar essas habilidades e o módulo especialista faz a análise das soluções para os problemas dos alunos.

O módulo de especificação de código é utilizado para garantir que a arquitetura seja independente da linguagem de programação usada. Este módulo armazena soluções de algoritmos para problemas de programação de forma independente da linguagem. O módulo de explicação, o módulo pedagógico e o módulo de estratégias instrucionais apresentada na Figura 14 correspondem ao módulo pedagógico da Figura 13. As estratégias pedagógicas de ensino contêm as estratégias relevantes empregadas pelo sistema enquanto o módulo de explicação é responsável por gerar as explicações reais de quaisquer erros cometidos pelos alunos. Assim, o módulo de aprendizagem e experiência são usados para melhorar o desempenho do tutor ao longo do tempo, aprendendo com as ações do aluno.

Porém, mesmo tendo sucesso em fornecer tutoria de um para um aos alunos, apenas alguns desses sistemas tutores inteligentes são usados em sala de aula devido ao custo elevado no desenvolvimento de tal sistema (PILLAY, 2003). A maioria dos STI's existentes é desenvolvido a partir do zero. A investigação de Murray e Woolf "(1992 *apud* TRUONG, 2003) mostrou que 100 horas de desenvolvimento é equivalente a uma hora de instrução. Pois, a construção de ferramentas de criação para proporcionar ambientes de desenvolvimento simples irá reduzir o número de módulos que precisar ser desenvolvidos a partir do zero e um sistema de tutoria inteligente modularizado permitindo assim que novos sistemas sejam criados reutilizando alguns componentes já existentes, reduzindo, assim, o tempo de desenvolvimento, Truong (2003).

3.3 Componentes de um Sistema Tutor Inteligente

3.3.1 Módulo de Domínio

O módulo de domínio é um dos componentes mais importantes de um Sistema Tutor Inteligente, porque é ele que contém o assunto que será ensinado pelo sistema. Além do mais, os outros módulos são geralmente construídos em torno do módulo de domínio, de modo que a representação usada aqui se torna muito importante.

Ao criar um STI para ensinar programação, um conceito importante é o fato de que a programação é uma habilidade prática que requer propiciar ao aluno uma ampla oportunidade de praticar para que eles tenham um bom proveito em seu aprendizado (GOMES, 2007). No entanto, apenas o fornecimento de exercícios de programação é insuficiente. Para que os alunos asseguram uma aprendizagem o *software* deve proceder analise do processo de desenvolvimento do algoritmo e verificar a capacidade do aluno em desenvolver o algoritmo para além do como mero mecanismo de resolução do problema de forma lógica, ou seja, se o aluno entendeu todo o processamento do algoritmo.

Este processo de análise automatizada é muito importante para os STI's que ensinam programação. No entanto, deve notar-se que existem outros sistemas informatizados que ensinam programação, e não são STI's, como os Sistemas Especialistas. Porém, esses sistemas não tem a preocupação de fazer uma

diferenciação sobre as necessidades especificas de cada aluno, uma vez que acaba nivelando todos os alunos em uma mesma categoria.

Embora a complexidade do modelo do estudante é detalhar ao máximo o conhecimento que se tem sobre o estudante, muitos modelos existentes insistem em perfeição e validez cognitiva. Se a meta for modelar o conhecimento do estudante de uma forma completa e precisa, o módulo do estudante pode ser considerado intratável (SELF, 1990).

3.3.2 O módulo pedagógico

O módulo pedagógico é um componente importante de um STI. Este módulo concentra-se em métodos para fornecer uma melhor aprendizagem para os alunos. Ele utiliza conceitos de diferentes disciplinas, principalmente Ciência Cognitiva e da Educação.

O módulo pedagógico em um STI para ensinar programação está concentrado em diferentes aspectos do ensino. Esta seção concentra-se no aspecto do módulo pedagógico que é relevante para essa pesquisa, o *feedback*.

3.3.2.1 Feedback

A análise de um algoritmo por um sistema, sem um retorno ao aluno, não é suficiente para que o processo de aprendizagem ocorra. Sendo assim, faz-se necessário que o sistema de análise de algoritmos forneça, durante a resolução do problema, um *feedback* apropriado e adaptado ao ensino para ajudar ao aluno na resolução do exercício. Esta ideia tem sido utilizada em muitos sistemas para fornecer diferentes tipos de *feedback* e outros métodos de apoio aos alunos que usam os STI no aprendizado de programação.

O *feedback* é um fator importante para facilitar a interação e ajudar na motivação dos alunos no processo de ensino e aprendizagem, quando eles se comprometem a sua prática. Muitos ambientes de programação têm funcionalidades internas para fornecer *feedback* em tempo real para os alunos. No entanto, um grande número dessas aplicações só ajuda os alunos com problemas relacionados a sintaxe e a semântica.

Fornecer *feedback* por si só não é suficiente para que os alunos aprendam. Muitos IDEs (*Integrated Development Environment*) são utilizados para programação e fornecem algum tipo de *feedback* através de mensagens de erro do compilador. No entanto, essas mensagens não são muito amigáveis e pode causar confusão para os iniciantes. Portanto, é essencial que qualquer tipo de *feedback* fornecido pelo sistema seja de fácil utilização (TRUONG, 2003). Por outro lado, o fornecimento em altos grau de *feedback* podem realmente resultar em um empecilho para o aluno diminuído a oportunidade de aprender (VANLEHN *et al.*., 1998). Portanto, o nível de realimentação é uma consideração importante ao fornecer as mensagens de *feedback* aos alunos.

A teoria histórico-cultural, uma das teorias da aprendizagem muito utilizado no campo da educação, concebe uma mediação que tem uma correlação com o conceito de *feedback* em um processo de aprendizagem que no âmbito do pensamento vygostkyano[3] trata-se da zona de desenvolvimento proximal (ZDP). Essa categoria é definida por Vygotsky como sendo essencial para a compreensão da aprendizagem. Em seu Livro “A Formação Social da Mente”, Vygotsky[4] (1991) conceitua a zona de desenvolvimento da seguinte forma:

> É a distância entre o nível de desenvolvimento real, que se costuma determinar através da solução independente de problemas, e o nível de desenvolvimento potencial, determinado através da solução de problemas sob a orientação de um adulto ou em colaboração com companheiros mais capazes. (VYGOTSKY, 1991, p.58).

Para Vygotsky, o processo de aprendizagem ocorre na zona de desenvolvimento proximal (potencial), por que é nessa zona que a interferência de outras pessoas (STI) é mais transformadora (OLIVEIRA, 2009). Assim, a zona de

---

[3] Para a teoria vygostyana o desenvolvimento cognitivo do indivíduo se dá por meio da interação social, ou seja, de sua interação com outros indivíduos e com o meio. Para essa teoria a aprendizagem é uma experiência social, mediada pela utilização de instrumentos e signos, de acordo com os conceitos utilizados pelo próprio autor.

[4] Lev Semenovich Vygotsky nasceu em 1896 na cidade de Orsha, na Rússia, e morreu em Moscou em 1934, com apenas 38 anos. Formou-se em Direito, História e Filosofia nas Universidades de Moscou e A. L. Shanyavskii, respectivamente. Por esses dados biográficos podemos perceber de início o pano de fundo que influenciou decisivamente a sua formação e o seu trabalho: a revolução russa de 1917 e o período de solidificação que se sucede. Vygotsky é um marxista e tenta desenvolver uma Psicologia com estas características.

desenvolvimento proximal refere-se ao nível de tarefas que são difíceis para um indivíduo realizar com o seu nível atual de conhecimento, mas pode ser realizado com a ajuda de um parceiro mais experiente (STI). O aprendizado ocorre melhor quando um tutor dá orientação nessa zona de desenvolvimento do aluno. Isso permite que o aluno amplie o seu conhecimento, alterando assim seu conhecimento sobre determinado assunto. Nesta perspectiva, entendemos que, é na zona de desenvolvimento proximal que os processos que não estão consolidados, requerem ou necessitam de uma ação externas para serem desencadeados.

Oliveira (2009), afirma que se tornaria sem feito tentar ensina amarrar sapatos a uma criança que sabe fazer tal ação e que a ação de um adulto, ao tentar ensinar essas habilidades a um bebê, também se torna sem efeitos, pois essas habilidades estão muito distantes do horizonte de suas funções psicológicas. Nesse sentido, a zona de desenvolvimento proximal de cada aluno é diferente. Isto significa que, algumas mensagens fornecidas pelo sistema podem ser adequadas para certos alunos enquanto pode ser inútil para outros. Com isso, um sistema que forneça um único nível de mensagens de *feedback* é eficiente quando estamos trabalhando com um amplo leque de alunos com personalidade e níveis e forma de aprendizagens diferentes.

Além da diversidade de tipos de *feedback* fornecido, quando um aluno comete um erro, alguns sistemas realmente dão um passo adiante ao corrigir os erros no programa dos alunos. O CPP-tutor (NASER, 2 e E. EIC (SYKES, 2007) são exemplos de tais sistemas. Ambos, primeiro fazem a pergunta ao aluno para determinar a correção dos erros, sugerida pelo sistema, antes de fazer a mudança real. No entanto, corrigir erros automaticamente pode atrapalhar o aprendizado dos alunos fazendo com que percam uma oportunidade de refletir e aprender por si mesmos (VANLEHN *et al.*., 1998).

3.3.3 Módulo do estudante

O módulo do estudante é um componente importante nos STI's, pois os mesmo tem o papel de individualizar as interações dos alunos com base em suas características. Os alunos são pessoas diversas e com muitas características diferentes, tais como os níveis de conhecimento, os estilos, as motivações, os

gostos e muito mais na forma de aprender. Todas estas características contribuem para que os seus métodos preferidos de aprendizagem devam, portanto, teoricamente, ser modelados de modo a individualizar a interação. Na prática, este é um problema muito difícil, devido a muitas razões (SELF, 1990). Uma vez que o foco desta pesquisa não é sobre o design do módulo de estudante, somente a característica que é mais diretamente relacionado à aprendizagem, o nível atual de conhecimento do aluno sobre o determinado assunto, é considerado neste trabalho.

O nível de conhecimento de um estudante a respeito de um determinado domínio, é difícil de mensurar. Porém, a fim de tornar esta medida mais precisa, geralmente é dividido os temas em tópicos, separando assim os componentes de conhecimento. O nível de conhecimento para cada um destes componentes é então considerado em vez de um nível global de conhecimento.

3.3.4 Módulo de comunicação

O módulo de comunicação é responsável pela forma de apresentação do material instrucional pelo sistema ao aluno, ou seja, a *interface*. Sabe-se que a uma boa *interface* é vital para o sucesso de sistema interativo, desta feita os sistemas tutores inteligentes não são exceção. Segundo Carvalho (2012), o usuário tem uma concepção que a *interface* é o próprio sistema, pois ao contrário, a *interface* é responsável, apenas, pela apresentação do material instrucional e pela monitoração do progresso do aluno através da recepção de suas respostas. Neste contexto, devem ser considerados a agradabilidade, facilidade, tempo de resposta e a monitoração (GIRAFFA, 2005 *apud* CARVALHO, 2012).

Para Carvalho (2012), a agradabilidade é essencial para evitar para evitar que aluno fique desmotivado e desinteressando. É necessário que o ambiente apresente uma riqueza de recursos na apresentação do material instrucional. Assim, o aluno deve contar com a facilidade e possiblidade de interagir facilmente com o ambiente. O tempo de resposta deve, evidentemente, permanecer dentro dos limites aceitáveis e o *feedback* deve ser realizada com o máximo possível de rapidez respeitando o tempo de resposta.

Devido ao grau elevado da interconexão entre os quatro maiores componentes de um STI, as técnicas usadas num módulo podem ser aplicadas aos

outros módulos. O uso de um modelo cognitivo para verificar erros do aprendiz ou desentendimento (módulo do estudante), tem o potencial não somente para apresentar conhecimento para o aprendiz (módulo de domínio), como faz o tradicional CAI ou a Instrução Baseada em Computador (CBI), mais também para comunicar conhecimento real usado durante a solução de problemas relevantes melhor que em situações abstratas (modelo instrucional).

As possibilidades de comunicar conhecimento (módulo pedagógico), baseada na prática atual, também fornece oportunidades para implementar estratégias educacionais que fornecem um fundamento para o aprendiz durante as fases iniciais da aprendizagem e logo, gradualmente, desvanecer-se como a perícia desenvolvida pelo aprendiz (GRAVIDIA e ANDRADE, 2003).

3.4 Sistema Tutor Inteligente para o Ensino no Domínio de Programação

Nesta seção serão apresentados brevemente alguns STI's encontrados na literatura para o ensino de programação. O conceito de Sistemas Tutores Inteligentes tem sido usado para ensinar assunto em muitos domínios. Como por exemplo, para o ensino sobre teoria de banco de dados, que é uma área que tem atraído o desenvolvimento de STI's, como por exemplo, o SQL-Tutor que é um dos mais bem-sucedidos STI de todos os tempos e que ensina aos alunos conhecimento referentes ao domínio de teoria e design de banco de dados, procurando ensinar aos alunos como escrever consultas SQL (MITROVIC & OHLSSON, 1999).

Outro STI é o *Access Personal Tutor*, que é um pouco diferente de outros tutores na medida em que funciona como um *add-in*[5] para o *Microsoft Access* para ensinar aos alunos como projetar formulários e relatórios (RISCO & REYE, 2009).

Muitos STI's, para o ensino de programação de computador, têm sido desenvolvidos em diferentes linguagens. Estes incluêm desde tutores de programação à companheiros virtuais de aprendizagem para auxiliar os alunos no desenvolvimento e resolução de algoritmos. Apesar dos avanços do desenvolvimento de STI's para o ensino no domínio de programação, nenhum deles ensina programação voltada para conceitos fundamentais de lógica.

[5] Na informática, é um *plugin* ou módulo de extensão.

No decorrer desse processo de desenvolvimento, diversos sistemas foram desenvolvidos para auxiliar alunos e professores no processo de aprendizagem da programação. Alguns desses sistemas são apresentados por Botelho, Nobre e Petry que passamos a descrevê-los sucintamente.

Lisp Tutor: STI foi desenvolvido na *Carnegie-Mellon University*, para acompanhar o aluno durante o processo de resolução dos problemas e transcrição do código. O sistema possui uma sequência de regras de produção que é repassada para o aluno à medida que ele vai resolvendo o problema, caso haja deficiências durante a produção, o aluno é informado e impedido de continuar até que seja resolvido aquele problema. Essa verificação do código é realizada quando o aluno aciona o tutor. A interação com o usuário é realizada através de um editor de textos e uma *interface* para o planejamento de programas que sejam mais complexos (BOTELHO, 2008).

SQL-Tutor: Sistema Tutor criado para o ensino da linguagem de banco de dados SQL. O sistema faz uma avaliação dos erros cometidos pelo aluno nas linhas de código e demonstra a correta solução. O sistema possui uma *interface* com o usuário; um módulo pedagógico, que é responsável por selecionar o problema, verificar os erros e gera as mensagens de erros apropriadas; e um módulo do estudante que é responsável por registrar o uso das restrições (NOBRE, 2002).

Algo-LC: STI desenvolvido na Universidade Federal de Santa Catarina - UFSC, que se diferencia dos outros Sistemas Tutores pelo fato de se ter um Companheiro de Aprendizagem, companheiro este que apoia o aluno durante a resolução dos exercícios, enviando mensagens o estimulando a verificar seus erros e corrigi-los e não demonstrando a resolução, como é feito em outros STI's (PETRY, 2005).

PROUST: é um dos sistemas mais antigos e famoso STI que se propõe ao ensino de programação, ele utiliza uma abordagem de indução. Neste sistema, os métodos de aplicação que são utilizados em programas são identificados e armazenados sob a forma de planos de programação. Estes planos incluem ambas as versões corretas e incorretas. As soluções esperadas para os exercícios são armazenadas como decomposições de objetivos constituídos por estes planos e

forma as várias interpretações da solução. Quando um aluno apresenta uma solução para um exercício, analisa-se contra as decomposições para tentar identificar qual interpretação o programa se encaixa (JOHNSON, 1990).

Um conjunto de regras de transformação também são mantidas para modificar o código para coincidir com os planos existentes. Heurísticas são usadas para determinar qual a interpretação de uma solução se encaixa mais de perto, a fim de determinar a intenção dos programas dos alunos. PROUST é capaz de analisar muitas soluções alternativas, gerando novas interpretações com base no programa que está atualmente sendo analisado. No entanto, com o aumento do número de planos de programação armazenada no sistema, torna-se mais difícil de identificar os planos reais utilizados pelo estudante. Isto é principalmente porque o sistema perde muito tempo para considerar todas estas soluções e decidir sobre uma provável interpretação da solução do aluno.

CPP-tutor: é mais recente que o *Proust*, e usa também a indução. Este tutor armazena uma solução correta para cada problema. Quando um aluno apresenta uma solução para um problema de programação, o sistema calcula uma distância de edição entre a solução do aluno e a solução correta usando a correspondência de padrões, a fim de identificar a intenção do estudante. O *Feedback* é então fornecida com base nessa análise (NASER, 2008).

Abaixo segue uma lista com outros STI’s utilizados nas mais diversas áreas de domínio:

Sistema	**Domínio**	***Feedback* e sugestões / caraterísticas especiais**	**Análise do programa**	**Seleção do Próximo exercício**
ELM-ART (WEBER & BRUSILOVSKY, 2001)	LISP	Hipermídia adaptativa	Identifica soluções semanticamente equivalentes utilizando transformação de plano e as regras de bugs.	Páginas que estejam em livro eletrônico são codificados por cores para indicar a adequadas para o aluno.
		Feedback sobre pedido		
		Identifica soluções completas e incompletas e fornece dicas		
		Vários níveis de sugestões		Estudante pode selecionar de forma diferente.
		Problema com base		
		OLM editável		
Prolog Tutor (HONG, 2004)	Prolog	*Feedback* sobre pedido	Compara a versão analisada de solução e de referência do programa, tanto do aluno usando um conjunto de técnicas comuns de programação Prolog.	Sistema seleciona um exercício com base no conhecimento atual do aluno.
		Programação guiada fornece modelos de técnicas de programação relevantes		
		Usa mensagens de erro baseados em técnicas de programação incorreta		
EIC (E. SYKES, 2007)	Java	*Feedback* sobre pedido	Análise baseada intenção usando árvores de decisão.	Sistema seleciona um exercício com base no conhecimento atual do aluno.
		Guias estudante para uma solução potencialmente único baseado na intenção		
		Correção automática de código se for caso disso		
AJATutor de programação (CORBETT, 2000)	LISP	*Feedback* imediato	Compara contra um conjunto de regras de produção	Conjunto pré-definido de exercícios apresentado no final de cada seção
	Prolog	Três níveis de sugestões		
	Pascal	OLM inspecionável		A área de domínio é apresentada com base no conhecimento atual do aluno.

Tabela 5 - Lista com STI's utilizados nas mais diversas áreas de domínio.

4 RACIOCÍNIO BASEADO EM CASOS

Com um trabalho do grupo de Roger Schank no início dos anos 80 do século XX, iniciaram-se os primeiros estudos sobre as técnicas baseadas em RBC, na Universidade de Yale, e as primeiras aplicações baseadas nesse modelo. E, especificamente em 1983, Janet Kolodner desenvolveu o primeiro sistema RBC, chamado *Cyrus*, o sistema continha as viagens e encontros do ex-secretário do estado dos Estados Unidos da América, Cyrus Vance, implementado como modelo de MOPs - *Memory Organization Packets* - de Schank, servindo de base para outros sistemas RBC (ABEL, 1996).

Porém, somente na primeira metade dos anos 1990 que a técnica de RBC atingiu sua maturidade como tecnologia de gerência e manipulação de conhecimento em alguns grupos na Europa e principalmente na Alemanha (WANGENHEIM; WANGENHEIM & RATEKE, 2013).

Segundo Abel (1996), esses trabalhos evoluíram rapidamente para aplicações de sistemas baseados em casos, principalmente nos domínios de Direito, Medicina e Engenharia, nessas aplicações busca-se resolver problemas de classificação, projetos, diagnósticos ou planejamento, especialmente em domínios onde o ser humano utiliza casos anteriores como base para a solução de problemas.

4.1 Definição de Raciocínio Baseado em Casos

O Raciocínio Baseado em Casos surgiu nos últimos anos como uma técnica de AI para solução automática de problemas. Conforme Wangenheim, Wangenheim e Rateke (2013), o RBC, devido a sua aplicabilidade de forma simples e direta, contém uma gama de espectros extremamente grande de problemas e situações reais, que justifica seu sucesso e rápida aceitação por quem desenvolve sistemas inteligentes em todo o mundo.

Um Sistema de Raciocínio Baseado em Casos – RBC, resolve problemas por adaptar soluções que foram utilizadas para resolver problemas anteriores. Segunda Wangenheim, Wangenheim e Rateke (2013), o RBC é uma técnica de IA que tenta simular o funcionamento do cérebro humano, buscando solucionar um novo problema através da recuperação e adaptação de casos passados armazenados na

base de conhecimento. O raciocínio analógico reconhece similaridades entre diferentes domínios e, a partir delas, pode gerar novos conhecimentos. Essa técnica utiliza diferentes cálculos de medida de similaridade como meio de mensurar o quão semelhante a um caso é de outro, considerando seus atributos e pesos associados a eles.

Wangenheim, Wangenheim e Rateke (2013), define quatro elementos básicos do ciclo de funcionamento de um sistema de RBC, que são:

- Representação do conhecimento: Aqui o conhecimento se apresenta como forma de casos que descrevem experiências concretas, porém outros tipos de conhecimento sobre o domínio de aplicação podem ser armazenados em um sistema de RBC como, por exemplo, casos abstratos e generalizados, tipos de dados e modelo de objetos usados como informação.
- Medida de Similaridade: o sistema tem que ser capaz de encontrar, na base de casos, um caso relevante para o problema atual e tem que ser capaz de responder à pergunta quando um caso relembrado for similar a um novo problema.
- Adaptação: Aqui, situações passadas representadas como casos dificilmente serão idênticas as dos problemas atuais.
- Aprendizado: diz respeito a um sistema que se mantenha atualizado e evolua, sempre que resolver um problema com sucesso, esse novo problema deverá ser armazenando na base de casos para que no futuro seja lembrado como um novo caso.

Conforme Wangenheim, Wangenheim e Rateke (2013) o RBC pode funcionar como um modelo cognitivo para se entender alguns aspectos do pensamento humano. Enquanto outras abordagens de Inteligência Artificial utilizam conhecimento genérico na forma de regras e roteiros, o RBC utiliza exemplos específicos concretos para representar o conhecimento, que é utilizado como base para resolução de outros problemas similares, conforme apresentamos na Figura 15.

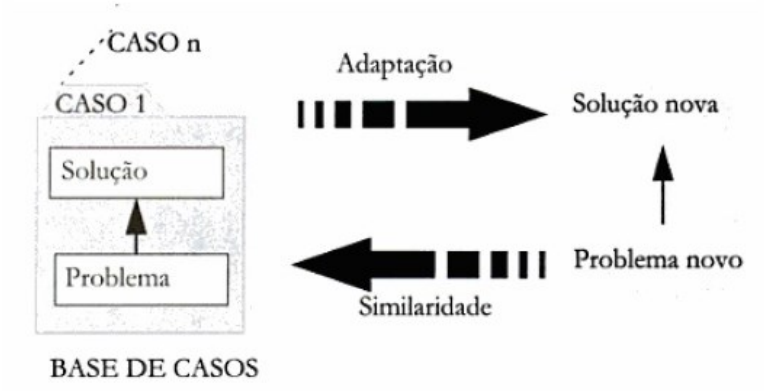


Figura 15 - Modelo do Raciocínio Baseado em Casos (WANGENHEIM; WANGENHEIM & RATEKE, 2013).

A ideia básica em um sistema de RBC é que, para um domínio particular, os problemas a serem resolvidos devem ser recorrentes e repetir-se com pequenas alterações em relação a sua versão original. Dessa forma, soluções anteriores podem ser reaplicadas também com pequenas modificações.

Essa é uma visão que se opõe aos sistemas de Raciocínio Baseado em Modelos - RBM - (do termo original em inglês, *Model-based Reasoning* - MBR), que utilizam modelo genérico de conhecimento do domínio, de forma que possa ser amplamente aplicável a diferentes problemas. Por exemplo, no caso de RBM, ao serem analisados diversos alunos com determinadas dificuldades em resolver determinado problema, tenta-se abstrair o que eles todos têm em comum, para construir o modelo do estudante, sem considerar as particularidades de cada aluno em específico. Já o RBC, em contraste, justamente enfatiza as instâncias das ocorrências da particularidade de cada aluno, e o conhecimento específico associado a cada situação particular.

O primeiro passo para utilizar uma solução já aplicada com sucesso anteriormente é determinar qual das experiências passadas mais se assemelha ao problema atual. Visando uma possível realização dessa comparação, é necessário que as experiências sejam analisadas e armazenadas de forma organizada. Os aspectos importantes dos problemas devem ser isolados, rotulados e ordenados em uma base de casos de tal forma que possam ser utilizados para comparar a situação atual com as anteriores armazenadas.

A revisão pode envolver outras técnicas de raciocínio, como o uso da solução proposta como ponto de partida para procurar uma nova solução, ou um ser humano poderia fazer a adaptação em um sistema interativo. A Retenção poderá então salvar o novo caso em conjunto com a solução encontrada.

Formalmente, um caso é composto por: uma descrição dos aspectos relevantes do problema que caracterizam aquela situação particular a ser resolvida, chamada de representação do caso; o contexto no qual o caso se insere, representado através dos índices do caso; a descrição da solução associada ao problema, na forma de um diagnóstico, uma classificação, uma sequência de ações, e; uma avaliação da solução empregada ao problema particular.

Na Figura 16 apresentamos o ciclo de funcionamento de um sistema de RBC, denominando de ciclo de solução de problema de um sistema RBC:

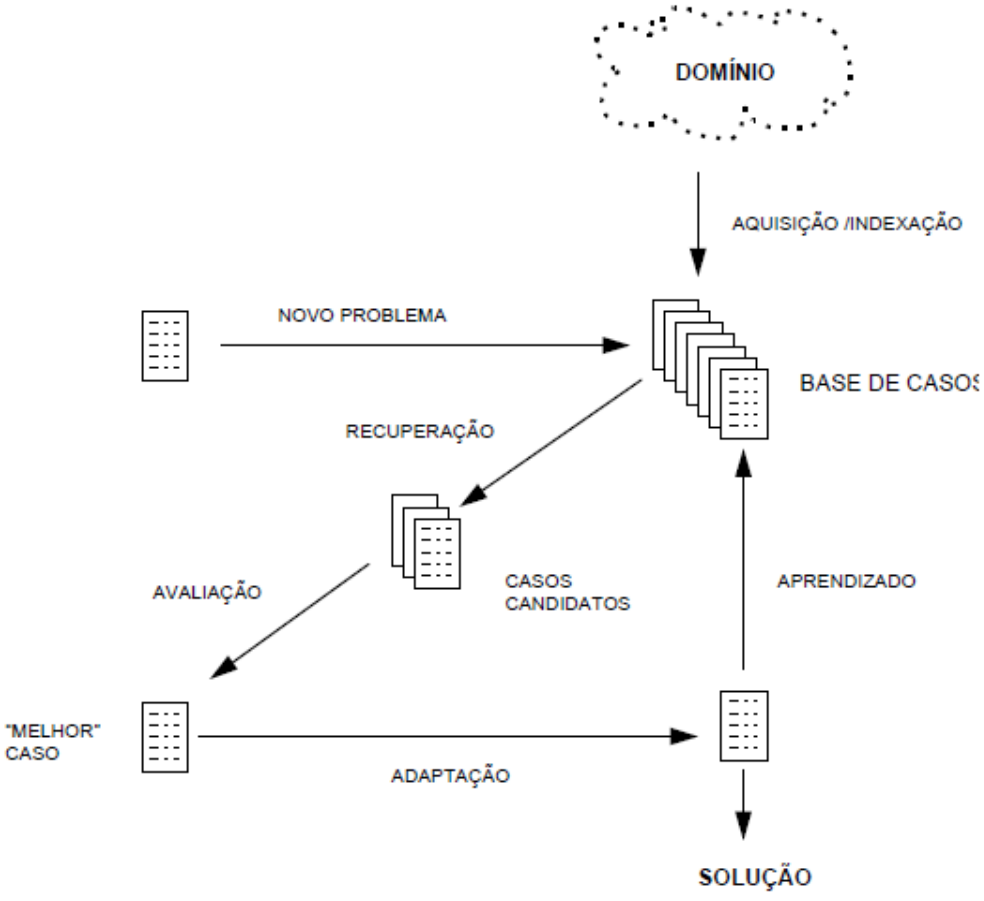


Figura 16 - Ciclo de solução de problema de um sistema CBR (WANGENHEIM; WANGENHEIM & RATEKE, 2013).

Assim, a cada novo caso um sistema de RBC deve reconhecer e definir o problema atual a fim de encontrar a nova solução. O sistema deve ser capaz de verificar a similaridade entre o caso atual e os casos armazenados na base de dados para, a partir deles, adaptar uma nova resolução para o problema dado.

Nessa perspectiva a qualidade de um sistema RBC depende de sua experiência, ou seja, depende do número de casos relevantes que farão parte da base de casos. Demonstramos, assim uma tabela com as etapas para construção de modelo de CBR:

Aquisição e representação	Selecionar os melhores dentro de um conjunto de casos disponíveis.
Indexação	Seleciona características da descrição dos casos que permitam recuperação eficiente dos casos relevantes ao problema.
Recuperação	Mecanismos para medir o grau de similaridade entre casos e problemas, seleção dos "casos candidatos".
Adaptação	Estratégias para adaptar a solução ou soluções dos casos recuperados; mecanismos para avaliar a qualidade da solução.
Aprendizado	Um problema resolvido e com solução validada pode ser acrescentado à base de casos.

Tabela 6 - Etapas para construção de modelo de CBR.

Desde 1990 o RBC tem crescido em um campo de inúmeros interesses, tanto no meio acadêmico quanto no comercial.

4.2 Representação o dos Casos

Um sistema de RBC é dependente da estrutura e do conteúdo de sua coleção de casos, que também pode ser denominada como estrutura de memória. O conhecimento em um sistema RBC é principalmente armazenado sobre a forma de casos. Para Kolodner (1993, *apud* SILVA, 2011), um caso é uma peça do conhecimento contextualizado representando um episódio ou experiência em que um problema foi parcialmente ou totalmente resolvido.

Os casos podem ser encontrados sob diversas formas e tamanhos, um caso pode ter diferentes conteúdos e representações dependendo da área de aplicação, por exemplo, venda de imóveis, atendimento a cliente. Conforme Wangenheim, Wangenheim e Rateke (2013), um caso representa tipicamente a descrição de uma situação (problema) juntamente com as experiências adquiridas durante a sua resolução (solução). Assim os autores afirmam que caso é composto dos seguintes componentes principais:

a) descrição do problema que foi resolvido ou a situação que foi compreendida: a descrição do problema apresenta informações sobre um problema que deve ser resolvido ou uma situação que deve ser interpretada, classificada ou compreendida. Essa descrição deve conter dados suficientes para que outros casos similares já ocorridos sejam identificados, ou seja, devem conter dados de forma que seja possível julgar a aplicabilidade de um caso a uma nova situação. A descrição do problema envolve a identificação dos objetivos a serem atingidos pela solução, das restrições impostas e dos atributos entre as partes do problema. Um caso armazenado na base de conhecimento deve incluir todas as informações consideradas para alcançar o objetivo específico.

b) solução descreve como o problema foi resolvido em uma situação particular: a solução de um problema descreve a forma como ele foi solucionado, apresentando conceitos e objetos utilizados para atingir os objetivos específicos do caso resolvido, levando em consideração as restrições e atributos. A estrutura da solução varia conforme aplicação e pode englobar o conjunto de passos seguidos no raciocínio, justificativas para as decisões tomadas, soluções alternativas, soluções inaceitáveis, além de solução em si.

c) Descrição do Resultado: a descrição do resultado descreve a consequência da aplicação da solução no problema inicial, incluindo um *feedback* do ambiente e uma interpretação do mesmo. O resultado pode englobar explicações sobre o sucesso ou fracasso da aplicação da solução, estratégias adotadas, se as expectativas foram alcançadas e formas de evitar o problema encontrado, etc.

Para que os casos estejam à disposição para serem reutilizados, eles são organizados e armazenados em uma base de casos. Geralmente uma base de casos contém experiências descrevendo os passos efetuados para resolver o problema em questão (SILVA, 2011).

O que é comum a todos os casos são que eles representam uma situação experimentada. Esta situação, quando relembrada mais tarde, forma um contexto no qual o conhecimento nela inserido é supostamente aplicável. Quando uma situação similar surge, aquelas decisões e o conhecimento que fez parte delas provêm um ponto de partida para a interpretação da nova situação ou para a solução do novo problema que esta situação coloca (WANGENHEIM, WANGENHEIM & RATEKE 2013).

Segundo Mendes (2012), existem diversas formas de representar casos, dependendo do enfoque do sistema RBC e dos objetivos perseguidos, como a representação na forma de atributo-valor, na forma orientada a objetos, através de redes semânticas ou em estruturas de árvores e grafos. A escolha da maneira de representação dos casos define a forma como o conhecimento será formulado.

4.3 Indexação de Casos

Ao determinar um caso, precisa-se identificar quais serão os índices para a indexação de um caso em sistema de RBC, devemos assegurar que um caso será acessado sempre que apropriado. Há que se considerar que a eficácia de um sistema de RBC, se dá quando ele é capaz de solucionar e recuperar casos relevantes de forma rápida e precisa. Mendes (2012) afirma que o entendimento sobre quando um caso deve ser recordado em situações futuras similares constituí um problema de indexação.

A indexação é vista como um problema de escolha de características que possam ser usadas como índices para os casos armazenados na memória, de forma que possam ser recuperados, quando necessário, identificando casos que possuem lições a ensinar. No entanto, a indexação pode também ser vista como um problema de organização da memória de casos, fazendo com que a recuperação seja realizada de forma precisa e eficiente. Wangenheim, Wangenheim e Rateke (2013), afirmam que, além de serem relevantes para uma meta de recuperação específica, bons índices para o RBC devem possuir as seguintes propriedades:

- serem fáceis de extrair dos casos armazenados na base de casos;
- serem usáveis e disponíveis como pistas na recuperação;
- categorizar casos na base de casos ao longo de algumas dimensões interessantes.

Hoje deparemos com várias formas de organização de casos na memória, entre as quais o modelo de estrutura linear, o modelo de estrutura hierárquica e também a organização pelo modelo de categoria e exemplar. Mendes (2012) conceitua essas formar de indexação da seguinte forma:

- estrutura linear: os casos são armazenados sequencialmente em uma lista, vetor ou arquivo e as características de cada caso são indexadas independente umas das outras. Esse tipo de armazenamento apresenta a vantagem de ser fácil aprender com casos recém-resolvidos, pois é simples e barato adicionar casos na memória, bastando inseri-los no final ou início da estrutura;
- estrutura hierárquica: faz com que somente um pequeno subconjunto dos casos seja considerado na recuperação. A hierarquia é geralmente obtida com a ajuda de métodos de agrupamento indutivo que provê uma forma de agrupar casos através de características que são partilhadas por um grande número de itens. Essa forma de agrupamento utiliza o conceito de árvores para separar os grupos, onde cada nó interno da rede mantém características partilhadas pelos casos abaixo dele, os nós sem aquelas características estão representados em nós irmãos ou em nós abaixo dos nós irmãos e os nós folhas são os próprios casos armazenados;

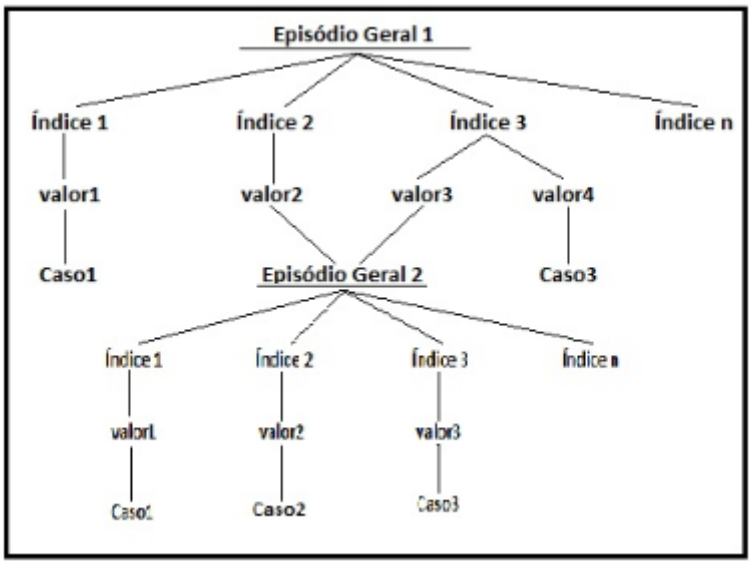


Figura 17 – Exemplo de arvores de características partilhadas (MENDES, 2012).

- a organização da memória pode também ser feita através do modelo de categoria e exemplar, onde os casos são referenciados como exemplares. A memória do caso é incluída em uma estrutura de rede de categorias, casos e ponteiros de índices. Cada caso é associado a uma categoria e um índice pode apontar para um caso ou categoria. Os índices são de três tipos: *links* de características apontando de descritores de problemas (recursos) para casos ou categorias (chamados de lembretes); *links* de casos apontando de categorias para seus casos associados (chamados de *links* exemplares) e *links* de diferença apontando dos casos para seus casos vizinhos que diferem em uma ou em um pequeno número de características. Uma característica é descrita por um nome ou um valor e os exemplares de uma categoria são ordenados de acordo com seu grau de prototipicidade na categoria.

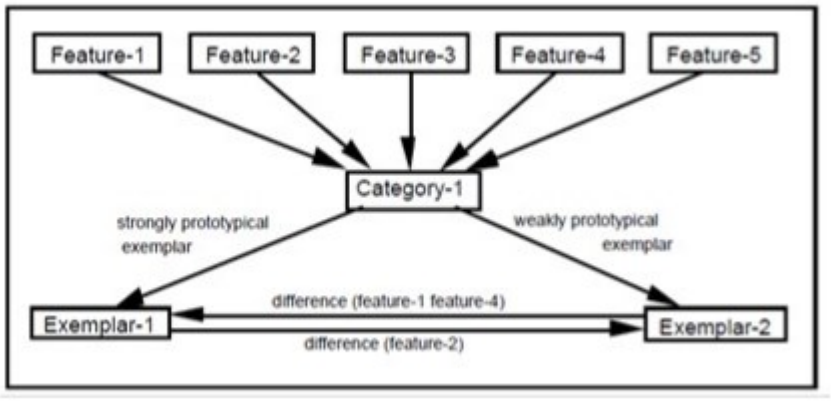


Figura 18 – Estrutura de Categoria, Características e Exemplares (AAMODT; PLAZA, 1994 *apud* Mendes, 2012).

Ainda segundo Mendes (2012) nessa organização de memória, as categorias estão inter-relacionados dentro de uma rede semântica, que contêm os critérios e os estados intermediários referenciados por outros termos.

4.4 Similaridade entre Casos

O conceito de medida de similaridade é a chave para a realização da busca por casos em sistemas de RBC. Uma medida de similaridade associa um valor numérico a um caso, expressando seu grau de similaridade com um novo caso. Essas medidas são baseadas em uma interpretação geográfica, na qual os casos são considerados pontos em um espaço n-dimensional. Cada dimensão corresponde a um atributo do conjunto de índices dos casos, que são um subconjunto dos atributos do conjunto da descrição do problema. O conceito mais simples de similaridade entre um caso e um caso recuperado é a medida da distância euclidiana entre os pontos no espaço n-dimensional correspondentes ao novo caso e ao caso recuperado, respectivamente (WANGENHEIM; WANGENHEIM & RATEKE, 2013).

O grau de similaridade entre os casos serve para determinar uma ordem parcial sobre os casos da base de conhecimento em relação à utilidade dos mesmos na resolução do problema que o sistema está tentando resolver, (WANGENHEIM; WANGENHEIM & RATEKE, 2013).

Hoje, basicamente, são utilizadas para determinar o grau de similaridade entre dois casos: as medidas de similaridade local e a global. No então, existem diversas funções de similaridade já definidas, tanto locais como globais.

4.4.1 Medidas de Similaridade Local

Wangenheim, Wangenheim e Rateke (2013), defini que a medida de similaridade local trata à semelhança dos casos em nível de atributos individuais e a define como a função descrita na Equação, conforme demonstramos a seguir,

$$sim(x_i, y_i) : (U \times U \rightarrow [0, 1]),$$

Equação 1 - Semelhança dos casos a nível de atributos individuais.

que determina a similaridade entre duas entidades de informação, onde U é o conjunto de casos representados no sistema.

Segundo Wangenheim, Wangenheim e Rateke (2013), as medidas de similaridade local são definidas conforme o tipo de dado do atributo, que podem ser numéricos, simbólicos, *strings*, entre outros. Além disso, essa medida é dependente do contexto da aplicação, variando conforme a aplicação RBC. Atualmente, existem várias formas de medidas de similaridade que consideram vários tipos de atributos.

4.4.2 Medidas de Similaridade Global

A medida de similaridade global mede a similaridade geral entre dois casos, ou seja, determina o quanto um dado caso é semelhante ao outro, considerando todos os seus atributos. (WANGENHEIM; WANGENHEIM & RATEKE, 2013) define a medida de similaridade global como a função descrita na Equação 2,

$$sim(x, y) : (U \times U \rightarrow [0, 1])$$

Equação 2 - Medida de similaridade global.

que determina a similaridade entre a pergunta e um caso sob consideração de todos os índices, onde U é o conjunto de todos os casos representados no sistema.

Dentre as técnicas de medidas de similaridades global podemos destacar a distância do vizinho-mais-próximo, que será a que utilizamos nesse trabalho. A distância do vizinho-mais-próximo utiliza cálculos bem simples e considera os casos como pontos em um espaço multidimensional, onde a dimensão do espaço é determinada pela quantidade de atributos dos casos. Dado que cada atributo assume um índice, a similaridade pode ser determinada através da distância espacial entre os pontos. A distância do vizinho-mais-próximo pode ser calculada através da Equação 3.

$$sim(Q, C) = \sum_{i=1}^{n} f(Q_i, C_i).$$

Equação 3 - Distância do Vizinho-mais-Próximo.

A função f determina uma medida de similaridade local entre os casos C e Q.

4.5 Recuperação dos Casos

Conforme Wangenheim, Wangenheim e Rateke (2013) o objetivo da recuperação de casos é encontrar um caso ou um peque no conjunto de casos na base de casos que contenha um a solução útil para o problema ou situação atual. Por exemplo, dada a descrição de um problema de erro em um algoritmo, um sistema de RBC deverá ser capaz de recuperar um caso descrevendo uma solução apropriada ao problema.

Segundo afirma Mendes (2012), na resolução de um novo problema, o sistema RBC recupera da memória o caso que se assemelha mais à nova situação e apresenta uma solução baseado no raciocínio do caso recuperado para auxiliá-lo na resolução. A tarefa de recuperação começa com uma descrição do problema e

termina quando é encontrado um caso prévio com melhor ligação. Mendes (2012) divide a tarefa de recuperação em três subtarefas que são a identificação das características do problema e a inicialização da ligação, que consiste nas tarefas de selecionar um conjunto de casos similares e selecionar o melhor entre eles, conforme explicita os passos que se segue:

- Identificar as características: identificar um problema pode envolver simplesmente à observação de suas entradas, mas, muitas vezes, uma abordagem mais elaborada é utilizada para entender o contexto do problema. Entender um problema envolve retirar ruídos dos descritores para inferir características relevantes, checar com outros valores de características que fazem sentido no contexto, gerar expectativas de outras características, etc. Outros descritores diferentes das entradas podem ser inferidos usando um modelo de conhecimento geral ou recuperando uma descrição de problema similar da base de casos e usando essas características como esperadas. A verificação de expectativas pode ser feita dentro do modelo de conhecimento (casos e conhecimentos gerais) ou solicitando ao usuário.
- Selecionar conjunto de casos similares: a seleção de um conjunto de casos correspondentes é feita usando os descritores do problema (características de entrada) como índice dos casos em memória de forma direta ou indireta. Existem três formas básicas de recuperar um caso: seguindo diretamente os ponteiros dos índices das características, procurando em uma estrutura de índice ou procurando em um modelo de conhecimento de domínio geral. Os casos podem ser recuperados apenas por características de entrada ou também pelas características inferidas da entrada. Casos armazenados na base que possuem ligação com todas as características são claramente bons candidatos para recuperação, mas casos que possuem uma fração de características com ligação também podem ser recuperados, desde que haja uma forma de avaliar quão similar ele é do problema. Avaliações de similaridade podem ser intensivas no conhecimento, tentando entender o problema profundamente analisando os objetivos e restrições. Outra opção é acrescentar pesos aos descritores

do problema de acordo com sua importância na caracterização do problema, durante a fase de aprendizagem.

- Selecionar melhor caso: a partir do conjunto de casos previamente selecionados, é escolhido o melhor caso. O melhor caso é determinado pela avaliação do grau de similaridade mais perto do caso apresentado. Essa avaliação é feita por uma tentativa de gerar explicações para justificar as características não idênticas, com base no conhecimento semântico da rede. A tarefa de seleção gera consequências e expectativas de cada caso recuperado e tenta avaliar as consequências e justiças as expectativas. Isso é feito através de um sistema de conhecimento de domínio geral ou com a ajuda do usuário. Posteriormente, os casos são classificados segundo algumas métricas, selecionando o caso que possui a explicação mais semelhante ao novo problema.

4.6 Reutilização dos Casos

Mendes (2012) afirma que a reutilização do caso recuperado como solução no contexto do novo caso foca dois aspectos: na diferença entre o caso passado e corrente e na parte do caso recuperado pode ser transferido para o novo. Para solucionar o problema corrente a tarefa de reutilização de casos utiliza as subtarefas de cópia e adaptação do caso anterior.

- Cópia: em tarefas simples de classificação as diferenças entre casos corrente e recuperado são abstraídas (relevando apenas as semelhanças) e a solução do caso recuperado é transferida para o novo caso. Esse tipo de reuso é trivial, no entanto outros sistemas devem levar em consideração as diferenças, exigindo um processo de adaptação das mesmas.
- Adaptação: existem duas formas principais de reutilizar casos passados: reutilização da solução do caso passado (reuso transformacional) e reutilização do método que construiu a solução do caso passado (reuso derivacional). No reuso transformacional a solução do caso passado não é diretamente a solução para o novo caso, mas existe algum conhecimento na forma de operadores transformacionais T que podem ser aplicados na velha solução transformando-a na solução do novo caso. Uma forma de

organizar esses operadores T é indexando-os em torno das diferenças detectadas entre o caso corrente e o recuperado. O reuso transformacional não se atenta em como um problema é resolvido, mas foca na equivalência de soluções, e isso requer um forte modelo dependente de domínio na forma de operadores transformacionais T juntamente com um regime de controle para organizar as aplicações operadoras. O reuso derivacional se atenta em como o problema foi resolvido no caso recuperado. O caso recuperado armazena informação sobre o método utilizado para resolver o problema, incluindo uma justificativa do operador utilizado, objetivos considerados, alternativas geradas, caminhos procurados que falharam, etc. O reuso derivacional utiliza o método do caso antigo e substitui no novo contexto. Durante a substituição alternativas de sucesso, operadores e caminhos serão primeiramente explorados, enquanto caminhos que falharam serão evitados; novos objetivos são perseguidos baseados no modelo do caso antigo.

4.7 Revisão de Casos

A tarefa de reutilização de casos pode gerar soluções corretas ou não. Quando a solução gerada é incorreta, o sistema tem a oportunidade de aprender com o erro ocorrido. A fase de revisão consiste nas tarefas de avaliar e reparar as soluções quando necessário. Em ambos os casos o sistema aprende com o novo caso resolvido (Mendes, 2012).

- Avaliação de Soluções: essa tarefa toma os resultados da aplicação da solução encontrada em um ambiente real (consultando um professor ou atuando em um ambiente real, que pode envolver passos externos ao sistema RBC). A avaliação das soluções em ambiente real pode levar algum tempo e o sistema marca esses casos como não avaliados. Outra forma de avaliação da solução é feita através da aplicação em programas de simulação habilitados a gerar uma solução correta.
- Reparação de Falhas: a reparação envolve a detecção de erros da solução corrente e a recuperação ou geração de explicações de soluções que não alcançam certas metas. Essas explicações podem ser

armazenadas em uma memória de falha como uma forma de prever possíveis deficiências, e posteriormente evitar e manipular erros. Uma segunda fase da revisão repara a solução, utilizando as explicações para modificar a solução de modo que falhas não ocorram.

4.8 Retenção de Casos

Para Mendes (2012) essa fase do processo retém o que é útil do novo problema resolvido na base existente de conhecimento. O aprendizado com o sucesso ou falha da solução proposta é disparado na avaliação ou possível reparação do caso. Essa tarefa envolve selecionar quais informações do caso armazenar, de que forma armazenar, como indexar esse caso para posterior recuperação em problemas similares e como integrar o novo caso na estrutura da memória.

- Extração: independente da forma como o problema foi solucionado o sistema RBC será atualizado. Se for resolvido usando um caso prévio, um novo caso será construído ou o caso anterior será generalizado englobando o presente caso. Se a solução utilizou outros métodos, um novo caso será construído. Em ambos os casos, é necessário escolher o que será utilizado como fonte de aprendizado e muita das vezes são utilizados os descritores e a solução do caso. No entanto, uma explicação ou outra forma de justificativa do por que dessa solução é apropriada ao problema pode ser armazenada juntamente com o caso. De maneira análoga, falhas podem ser extraídas e retidas no sistema afim de evitar erros futuros.
- Índice: o problema de indexação é bastante focado em sistemas baseados em casos e envolve a decisão de que tipo de indexação utilizar para futuramente recuperar casos e como estruturar o espaço de procura de índices.
- Integração: esse é o último passo na atualização da base de conhecimento. Se não foi construído um novo caso e índice, esse passo é o principal na retenção. Modificando a indexação de casos existentes os sistemas RBC aprendem a se tornar avaliadores de similaridades, pois a seleção de índices é uma parte importante do aprendizado do sistema.

Índices fortes ou importantes para um caso ou solução em particular são ajustados de acordo com o sucesso ou falha do problema de entrada. Características julgadas relevantes para a recuperação são associadas ao caso com maior peso, enquanto que características menos importantes são ligadas fracamente a ele. Dessa forma, a estrutura de índice tem o papel de ajustar e adaptar o processo de uso da memória.

5 STI BLOP PARA O ENSINO DE LÓGICA DE PROGRAMAÇÃO

O STI BLOP é uma ferramenta para dá suporte ao aluno a construir seus primeiros algoritmos e com isso adquirir conhecimento no domínio de lógica de programação. O STI utiliza uma linguagem visual em blocos que permite criar programas em blocos estruturados. Isso é feito principalmente através do fornecimento de exercícios de programação para os alunos resolverem. A fim de que se tenha um bom proveito no ensino da disciplina, é necessário que o sistema de análise das respostas dos algoritmos dados pelos alunos forneça um *feedback* construtivo e contribua para o seu processo de ensino.

Este protótipo faz parte de um projeto, cujo a pretensão foi construir um *software* para auxiliar alunos e professor no processo de ensino e de aprendizagem de algoritmos. Neste *software* o aluno, através de um Sistema Tutor Inteligente, é induzido a produzir respostas para questões de algoritmos.

5.1 Descrição da Arquitetura do STI BLOP

O Sistema Tutor Inteligente BLOP é projetado para ensinar lógica de programação para alunos iniciantes em cursos da área de computação que aborde essa disciplina. Assim, o sistema disponibiliza, principalmente, exercícios de programação para serem resolvidos pelos alunos. O sistema ajuda aos alunos resolverem novos problemas, fornecendo como *feedback* soluções que foram usadas para os problemas anteriores e semelhantes. Portanto, para um sistema ensinar o conhecimento para o qual foi projetado para ensinar, de forma eficaz, faz-se necessário que ele analise as respostas dadas pelo aluno e forneça *feedback*, baseado em casos anteriores, relevantes e construtivos.

Uma das maiores dificuldades em se trabalhar com sistema para análise de exercício de algoritmo é que um exercício de programação apresenta várias possibilidades de soluções corretas para um mesmo problema. Para exemplificar, tomamos o exercício de programação descritos a seguir, embora este seja um exercício muito simples, o programa pode ser escrito de muitas maneiras. Vejamos a apresentação:

Problema: Escrever um programa chamado "soma", em que a soma de todos os números inteiros entre 1 e um número específico N. Por exemplo, se a N for atribuído o valor 10, então a soma dos números de 1 a 10 é de 55.

Figura 19 mostra a especificações do programa. Este programa requer o uso de uma estrutura de repetição “se...faça”. A saída será igual a: soma = 55.

Figura 19 - Linguagem visual problema soma.

Porém, pode-se reconhecer que existem várias possibilidades para as respostas dos alunos e o sistema não pode simplesmente listar respostas corretas como mensagens de *feedback*, mas tem que, baseado em casos anteriores, sugerir possibilidades de formas de resolução do problema e apresentar para alunos. Por exemplo, a Figura 20 mostra como o aluno poderia escrever de forma diferente o mesmo problema com uma condição para entrada na estrutura de seleção:

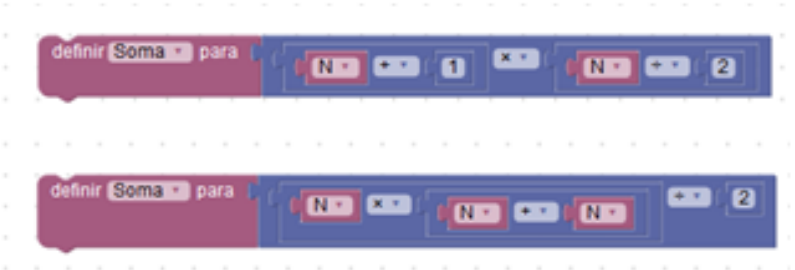


Figura 20 - Exemplos de resposta em linguagem visual.

Ambas as respostas são completamente corretas e o sistema precisa reconhecer esses tipos de respostas e não apenas responder de volta para o aluno indicando uma falha. Testar a correção de um programa não é uma tarefa fácil, esse resultado não pode ser alcançado apenas dando um conjunto de respostas fixas, por isso o STI BLOP é um modelo desenvolvido para analisar as atividades dos alunos e apresentar como *feedback*, através de experiências similares já vividas, atividades armazenadas na base de casos para que sirva como modelo para reflexão do aluno para resolver um novo problema que tenha aspectos similares.

Com isso podemos perceber que análise de programa fazendo-se a comparação linha por linha não é método muito eficaz de comparação de similaridade entre algoritmos. Pois, se uma única solução for mantida, muitos desses programas serão identificados como incorreto, apesar de resolver o mesmo problema proposto.

Na Figura 21, apresentamos o modelo de arquitetura de processo de verificação de respostas fornecidas pelos alunos para o STI BLOP.

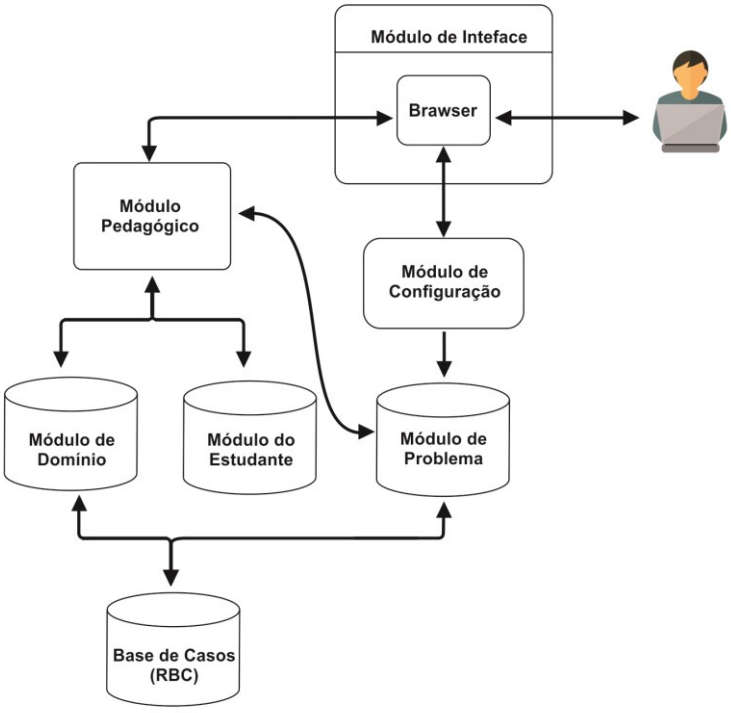


Figura 21 - Arquitetura Geral do STI BLOP.

A solução do aluno é analisada contra um conjunto de programas corretos armazenados na base de casos. Devido à complexidade envolvida com a análise dos algoritmos, é necessário restringir ao STI BLOP um pequeno subconjunto da linguagem de programação visual com a qual trabalhamos. A área de foco envolve a seguinte lista de conceitos básicos em linguagem visual em bloco:

a. variáveis (declaração e uso local e global);

b. operadores; e

c. estruturas de loops.

Este sistema armazenará várias soluções para comparação entre os problemas já armazenados e os novos problemas propostos. Quando um aluno apresentar uma solução para um dado problema de programação, o sistema verificará a similaridade entre as soluções proposta pelo o aluno e as soluções armazenadas na base de casos, a fim de identificar possíveis erros na resolução dos exercícios realizados pelos alunos. Assim, o *feedback* é, então, fornecido com base na análise de problemas anteriores. RBC pode ser visto como um ciclo de cinco tarefas seguintes.

1. dado um novo caso, recuperar casos semelhantes a partir da base caso;
2. comparação parcial dos casos da base com o problema atual;
3. ordenação dos casos selecionados na base de acordo com o valor da similaridade;
4. avaliar a solução e revê-la com base em quão bem ela funciona;
5. decida se deseja manter este novo caso na base de casos.

Será construída uma base de casos de registros com a seguinte estrutura: Dado um problema (P), há um número N de soluções (S1, S2, ..., SN), com um número (m) de categorias de respostas corretas classificáveis (R1, R1, ..., Rm). Para cada categoria de resposta correta existe um número finito (f) de *feedback* adequados (F1, F2, ..., Ff). O objetivo é recuperar, através da técnica de RBC, o caso “mais similar”, ou o grupo de casos mais similares, armazenados na base de casos, em relação ao problema atual. Assim, em RBC a recuperação não depende somente de um caso propriamente dito, mas também de outros, possivelmente

melhores, existentes na base de casos. E segundo Main e Shiu (2001), há um grande benefício no uso do RBC, que é um sistema que pode ser criado com uma pequena ou limitada quantidade de experiência – casos - e de forma incremental sendo desenvolvida ao se acrescentar mais casos à base de dados, assim que estiverem disponíveis.

Conforme Wangenheim, Wangenheim e Rateke (2013), um Sistema Tutor Inteligente baseado em RBC para ser aplicado no ensino e aprendizagem, tem que demonstrar capacidade e ser capaz de gerar soluções para problemas de forma automática, com base em seu conhecimento de domínio. Isto permite ao sistema analisar soluções complexas e incomuns aos problemas, reconhecer e explicar erros. Por outro lado, a representação do comportamento do usuário em modelos de aprendizado permite ao sistema adaptar-se dinamicamente às necessidades do estudante.

Seja qual for o método de análise utilizado, o objetivo final de um sistema projetado para ensinar programação deve identificar com o processo desenvolvido da lógica elaborada pelo aluno e apresentar possibilidades, quando incorreta, de formas de alteração para que o aluno alcance êxito na resolução do problema. Os diferentes métodos de análise são bem-sucedidos para alcançar isso, em diferentes graus. Ao conceber o módulo do domínio de um STI para ensinar programação, é necessário selecionar o método adequado para o sistema proposto.

Vale ressaltar que o STI BLOP é indicado para alunos iniciantes na resolução das atividades práticas das disciplinas de Introdução a Lógica de Programação, mais especificamente para alunos dos cursos relacionados à área de informática que faz uso dessa disciplina.

5.2 Especificação

Nesta seção apresentaremos os artefatos criados para o desenvolvimento do ambiente. Serão apresentados os requisitos funcionais e não-funcionais, a descrição dos atores e o projeto arquitetural do ambiente.

Sommerville (2007) os requisitos de um sistema são as descrições dos serviços fornecidos pelo sistema e as suas restrições operacionais. O levantamento

dos requisitos é uma etapa que compreende os problemas que serão aplicados no desenvolvimento de um *software*. Um requisito é uma condição ou capacidade que tem como objetivo a ser alcançada por um sistema ou componentes de um sistema para atender um contrato, ou um padrão de especificação ou outros documentos formalmente impostos (Maciaszek *apud* BEZERRA, 2006).

Os requisitos aqui expostos, para o desenvolvimento do sistema são requisitos básicos necessários para garantir o funcionamento do mesmo.

5.2.1 Requisitos Funcionais

Iremos dividir os requisitos funcionais do sistema em requisitos funcionais do professor (administrador) e requisitos funcionais do aluno. São funções detalhadas do comportamento esperado de um sistema, mostrando como ele reage a entradas específicas e seu comportamento em determinadas situações. Em um sistema servem para descrever essas funções ou até mesmo serviços que se espera de um sistema, podendo definir funções detalhadas como entradas e saídas (SOMMERVILLE, 2007).

RF1 **Requisitos funcionais do professor (administrador)**: O ambiente deve permitir ao professor (administrador) do sistema o gerenciamento e organização das atividades e todo o processo de cadastramento, remoção, inclusão e manutenção do sistema.

RF1.1 o sistema deve permitir ao administrador adicionar, remover e manter atividades de programação no sistema;

RF1.2 o sistema deve permitir ao administrador a visualização e alterações das informações sobre as atividades ensinadas pelo sistema;

RF1.3 o sistema deve permitir que o administrador possa visualizar relatório de desempenho individual do aluno;

RF1.4 o sistema permite ao administrador salvar novos casos que estarão com o status de pendente na base de casos;

RF2 **Requisitos funcionais do aluno (interação com o sistema)**: O sistema deve permitir a manutenção dos problemas, visualização e o reuso de suas soluções.

RF2.1 o sistema deve permitir manter e adicionar os problemas de programação.

RF2.2 o sistema deve permitir ao aluno visualizar as informações sobre o problema que está sendo resolvido.

RF2.3 o sistema deve permitir a visualização das informações sobre os problemas que foram resolvidos.

RF2.4 o sistema deve permitir que um determinado problema seja resolvido.

RF2.5 o sistema deve permitir ao aluno que a solução de um determinado problema seja submetida para avaliação.

RF2.6 o sistema deve permitir que a solução para um determinado problema seja avaliada, possibilitando a emissão dos devidos *feedback*.

RF2.7 o sistema deve possibilitar a visualização dos *feedback*s emitidos sobre a solução para o problema.

RF2.8 o sistema deve permitir que o problema resolvido seja executado pelo interpretador.

RF2.9 o sistema deve permitir a recuperação dos problemas similares armazenados na base casos - O ambiente deve recuperar da sua memória uma lista de casos similares.

5.2.2 Requisitos não-Funcionais

São restrições sobre os serviços ou as funções oferecidas pelo sistema, podendo está relacionado à propriedade de sistema. Assim ele pode expressar tempo de resposta, confiabilidade e espaço em disco. Ainda existe a alternativa até podendo definir restrições para o sistema (SOMMERVILLE, 2007). Os requisitos não funcionais não descrevem as funções especificas do sistema.

Requisitos não funcionais do sistema:

- A solução envolveu somente tecnologia *open source*;
- Os perfis de acesso ao sistema são:
    - Administrador: tem permissão para efetuar todas operações;
    - Aluno: tem permissão para acessar atividades, enviar e manter respostas no sistema;
- O sistema utilizará o banco de dados *PostgreSQL*;
- Para o desenvolvimento das aplicações de RBC utilizaremos a plataforma *jCOLIBRI*;
- Foi utilizado a última versão do *Blockly* encontrada no *GitHub* em 21/10/2015.
- Dimensão da base de casos: A base de casos deve oferecer ao aluno situações que englobem, além das listas de exercícios indicadas pelo sistema, problemas extras para que o aluno possa praticar. Para tanto, é necessário ampliar a base de dados.
- Tempo de resposta do sistema: O sistema deve limitar seu tempo máximo de resposta em 30 (trinta) segundos para as atividades que necessitem da comunicação entre o cliente e o servidor.
- *Interface* de fácil uso: A *interface* do sistema deverá possibilitar a consulta e utilização dos casos, dos problemas resolvidos e dos *feedbacks* com no máximo 3 (três) cliques.

5.2.3 Atores do sistema

O STI BLOP possui 2 (dois) usuários distintos:

- **O professor (administrado)**: este ator utiliza o sistema para adicionar, remover ou modificar os problemas, cadastrar os alunos da disciplina, corrigir as soluções dos problemas propostas pelos alunos e manter os casos com o status de pendente.
- **Alunos**: interagem com o sistema para resolução dos problemas de programação, *feedback* com o sistema, visualização e envio de soluções corretas para serem armazenada ao sistema.

Nas Figuras 22 e 23, estrão ilustrado os diagramas de casos de usos que descrevem a interação entre os atores e os requisitos funcionais definidos:

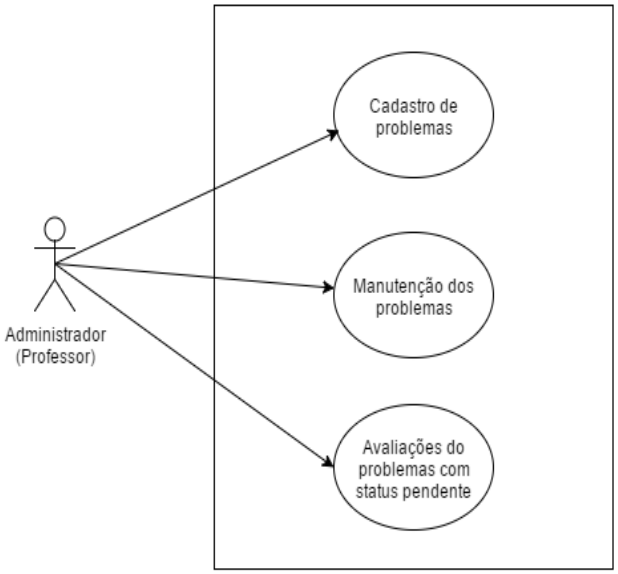


Figura 22 - diagramas de casos de usos Administrador/Professor.

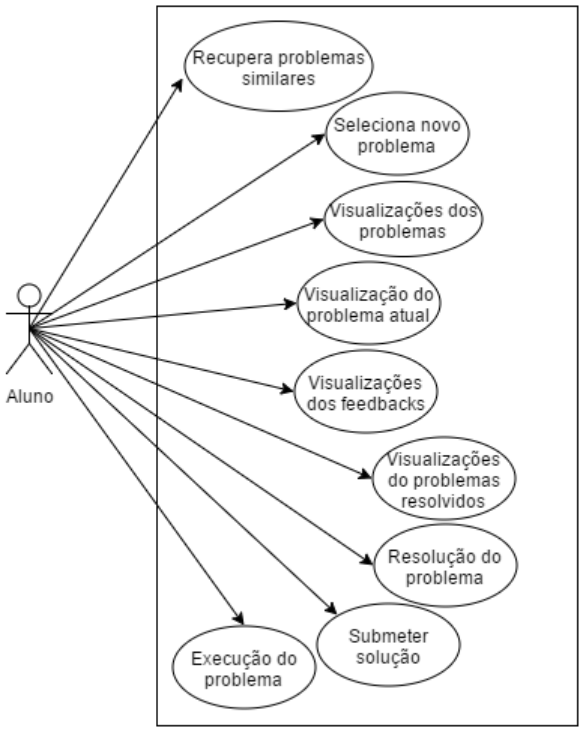


Figura 23 - Diagramas de casos de usos Aluno.

Dentro desse contexto, o lado cliente é dividido em 2 (duas) visões de acesso: a visão do aluno e a visão do professor.

5.3 Recuperação e Adaptação de Casos

5.3.1 Recuperação de casos

Na etapa de recuperação de casos na técnica de RBC o objetivo principal é recuperar o caso “mais similar”, ou o conjunto de casos mais similares, armazenados na base de casos, em relação ao problema ou situação atual. O processo de recuperação de caso utilizada nesta pesquisa, foi a métrica do vizinho-mais-próximo, onde para cada atributo pode-se obter um resultado de medida da distância entre o novo problema e os casos já registrados, possibilitando ainda a cada atributo possuir um peso diferenciado e determinando a importância do atributo para os resultados.

O cálculo utiliza o valor da similaridade entre os atributos definidos para a consulta e multiplica pelo respectivo peso, calculando o somatório de todos os atributos.

Segundo Wangenheim, Wangenheim e Rateke (2013), problemas similares possuem soluções semelhantes. Assim um caso é útil se ele é similar ao problema atual. A similaridade tem por objetivo selecionar casos que possam ser facilmente adaptados para o problema em questão.

Para esse autor, o processo de recuperação pode ser dividido nos seguintes passos:

1. descrição do problema/situação atual;

2. busca na base de casos;

3. comparação parcial dos casos da base com o problema atual;

4. ordenação dos casos selecionados na base de acordo com o valor da similaridade.

O algoritmo usado para calcular a similaridade depende da estrutura utilizada nos índices e de como os casos foram representados. Existem vários métodos para

cálculo de similaridade. Watson (1994) cita alguns métodos conhecidos: vizinho-mais-próximo, indução, indução guiada por conhecimento e recuperação de modelos. Porém, iremos nos focar aqui na técnica do vizinho-mais-próximo, que será utilizada para o desenvolvimento desse trabalho.

A técnica do vizinho-mais-próximo avalia os casos armazenados na base e tenta determinar o quanto cada caso armazenado é semelhante em relação ao caso de entrada. Para avaliar esta semelhança é usada uma soma ponderada que leva em conta os atributos do caso. A maior dificuldade é saber qual o será o peso dos atributos de um caso. Segundo Watson (1994) esta técnica apresenta limitações relacionadas à correta identificação do melhor caso e o tempo de recuperação deste melhor caso: o tempo de recuperação aumenta linearmente em relação ao número de casos. Por este motivo, esta técnica é usada em bibliotecas de casos relativamente pequenas.

A técnicas do vizinho-mais-próximo é, talvez, a tecnologia mais amplamente utilizada em sistemas de CBR, uma vez que é fornecida pela maioria das ferramentas CBR (WATSON, 1994). Utiliza-se nessa técnica uma métrica para calcular a similaridade entre um caso atual (novo caso) e um caso anterior (caso passado). Essa similaridade é dada pela somatória da função de proximidade entre os atributos que representam os casos. Essa função pode ser representada pela distância espacial entre os atributos em um espaço multidimensional. Para garantir a prevalência de um atributo em relação ao outro, multiplica-se a função por um peso adotado por cada atributo.

A normalização do resultado calculado é determinada através da divisão do valor total da similaridade pela soma total dos pesos determinados pelo usuário. Assim, a similaridade entre dois casos pode ser representada através da equação:

$$Sim\ (T,S) = \frac{\sum_{i=1}^{n} f\ (T_i, S_i) x\ W_i}{\sum_{i=1}^{n} w_i}$$

Equação 4 - Similaridade entre dois casos.

Significado de cada variável:

T = é o novo caso;

S = são os casos existentes na base de casos;

n = é o número de atributos em cada caso;

i = é um atributo individual de 1 a n;

f = é uma função de similaridade para o atributo i em casos T e S, e;

w = é o peso do atributo i.

A maioria das ferramentas RBC utiliza algoritmos como este. Este cálculo é repetido para todos os casos, na base de casos para classificar casos por semelhança com o novo caso. Algoritmos semelhantes a este são usados pela maioria das ferramentas de RBC para executar a recuperação do vizinho-mais-próximo. Semelhanças são, geralmente, normalizadas para cair dentro de um intervalo de zero a um (onde zero é totalmente diferente e um é uma correspondência exata) ou como uma percentagem de similaridade, onde cem por cento é uma correspondência exata (WANGENHEIM; WANGENHEIM & RATEKE, 2013).

Utilizaremos para essa pesquisa a escala da faixa de peso, para medir a similaridade de casos, desenvolvida por Urnau, Kipper e Frozzar (2014) a faixa escolhida pelos especialistas foi de “0” até “10” para o fator (peso) que pode ser definido para cada atributo do caso. A definição desta variação ocorreu a partir da escolha feita pelos especialistas humanos. Para o fator peso na fórmula do vizinho-mais-próximo , o critério pode ser definido livremente. O que a fórmula matemática exige é que seja um algarismo numérico, podendo ser representado de “0,0” até “1,0”, de “0” até “100” ou “0” a “1000”, entre outros. Três fatores foram decisivos para determinar a faixa do valor a ser atribuído ao peso. O primeiro fator é que, os valores entre “0” até 10 permitem representar adequadamente a faixa de interesse para

cada atributo, qualificando este interesse entre (Nenhum, Pouco, Parcial, Bastante e Total) conforme figura 24, abaixo:

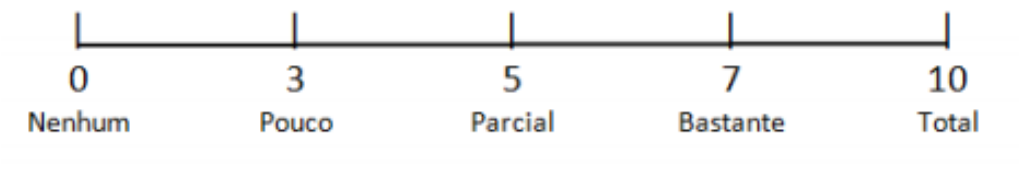


Figura 24 -Faixa de interesse para cada atributo (URNAU; KIPPER & FROZZA, 2014).

O segundo fator diz respeito ao produto final da equação, com os valores variando entre zero e dez. O resultado da fórmula não retorna um valor muito alto e extenso. E o terceiro fator é que nesta faixa de valores, a discrepância entre os resultados finais permite uma oscilação que melhor representa as similaridades ou dissimilaridades entre os casos, sendo mais condizente com as realidades. Para outros valores testados, como “0,0” até “1,0”, a discrepância entre os valores foi considerada pouca, causando a interpretação que todos os casos recuperados eram similares. Já com os valores de “0” a “100” ou “0” a “1000”, os resultados testados eram extensos e uma discrepância grande, causando a interpretação que todos os casos recuperados não tinham similaridade. Assim justificando a melhor escolha pela faixa entre os valores “0” até “10” Urnau, Kipper e Frozzar (2014).

A tabela a seguir demonstra o valor encontrado para um caso já armazenado na base de dados. O sistema realiza a soma da multiplicação de todos os atributos, dividindo pelo somatório do valor dos pesos informados pelo usuário. Conforme demonstrado na última linha da tabela. Determinando assim que a similaridade entre o caso buscado e o caso armazenado na base de casos é igual a 3.

Atributos	Peso	Caso buscado	Caso 1	Similaridade local
Declaração de variáveis	3	Definir...para	Definir...para	5
Loops	10	Repita...faça	Se...faça	5
Texto	10	Definir soma para	Definir soma para	1
Operadores	8	Lógico	Matemático	1
Valor da Similaridade Global: Caso 1 (3*5 + 10*5 + 10*1 + 8*1) / (3+10+10+8) = 15 + 50 + 10 + 8 + 0/31 = 3				

Tabela 7 - Similaridade entre o caso buscado e o caso armazenado na base de casos.

O cálculo da similaridade é realizado para cada atributo escolhido pelo usuário, sendo percorrida toda a base de casos que é dividida por nível ou categoria. Realizando-se o cálculo para todos os casos existentes em cada nível, o resultado é apresentado levando em consideração a ordem de maior similaridade encontrada entre os casos de uma mesma categoria.

5.3.2 Adaptação de Casos

A princípio a forma de adaptação seguirá os seguintes critérios: o sistema armazenará todos os casos, considerados como corretos, que forem avaliados para uma possível reutilização. O novo caso será inserido com um status de pendente, o que servirá para diferenciá-lo de casos que já foram aprovados, e estão sendo utilizados na base de conhecimento como base para as consultas. O administrador da base de conhecimento fará a avaliação dos novos casos inseridos através de uma consulta de casos pendentes, onde serão listados somente os casos com o status de não resolvido.

O usuário do sistema pode será habilitado a adicionar ao banco de casos aqueles casos que julgar necessário, porém os mesmo casos ficam disponíveis para avaliação do professor (administrador) sistema com o status de pendente até que o

mesmo seja alterado e novo caso passe a ser utilizado na base de conhecimento como base para consulta.

5.4 Modelo do STI BLOP

O STI BLOP é um sistema baseado na *web* que poderá ser acessado através de um navegador. Para utilizar o sistema, cada aluno deve criar uma conta de usuário com *login* e senha. Quando um aluno fizer *login* pela primeira vez, será solicitado que realize um pré-teste para medir seu conhecimento atual sobre o domínio de lógica de programação. O pré-teste é um conjunto de questões de múltipla escolha, onde o aluno poderá deixar em branco as questões das quais não sabe a resposta. Sendo possível até mesmo não responder todas as perguntas que não tem conhecimento sobre o assunto.

Uma vez que concluído o pré-teste, o aluno é direcionado para a página de seleção de exercícios. Esta página também será exibida no início de cada sessão seguindo uma ordem de nível desde o pré-teste. O aluno escolhe um exercício para tentar resolver e, em seguida, insere o código em linguagem visual com a solução do exercício como mostramos na Figura 25. Quando solicitado, o sistema fornece *feedback* para o usuário. Ao aluno, também é permitido abandonar o exercício atual e retornar para a página de seleção de exercício a qualquer momento.

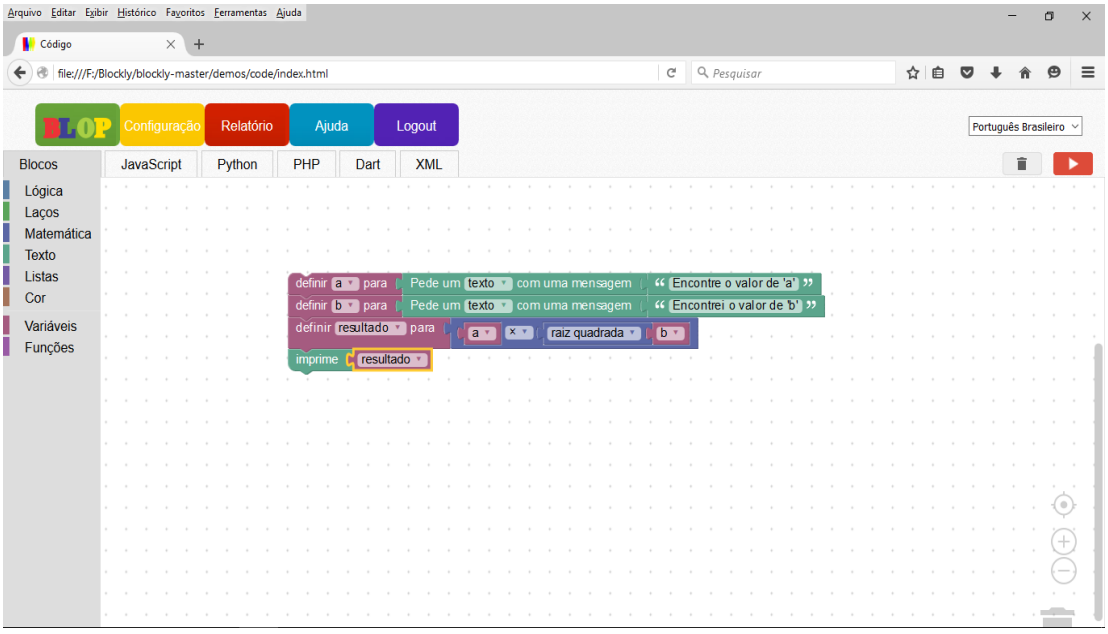


Figura 25 - Página de resolução de exercícios.

Na parte superior da página o sistema exibe um *menu,* como demostrado na Figura 26. Nesse *menu*, é permitido escolher entre algumas opções. O aluno pode sair do sistema ou alterar as configurações do nível de exercícios e senha. O *link* "ajuda" traz algumas páginas de informação sobre como usar o sistema. Também será permitido, ao aluno, visualizar um relatório individual de desempenho, através de uma página que exibirá o seu conhecimento atual sobre o domínio dos temas abordados pelos STI BLOP, o relatório será aferido pelo sistema, com base nas resoluções de atividades realizadas pelo aluno.

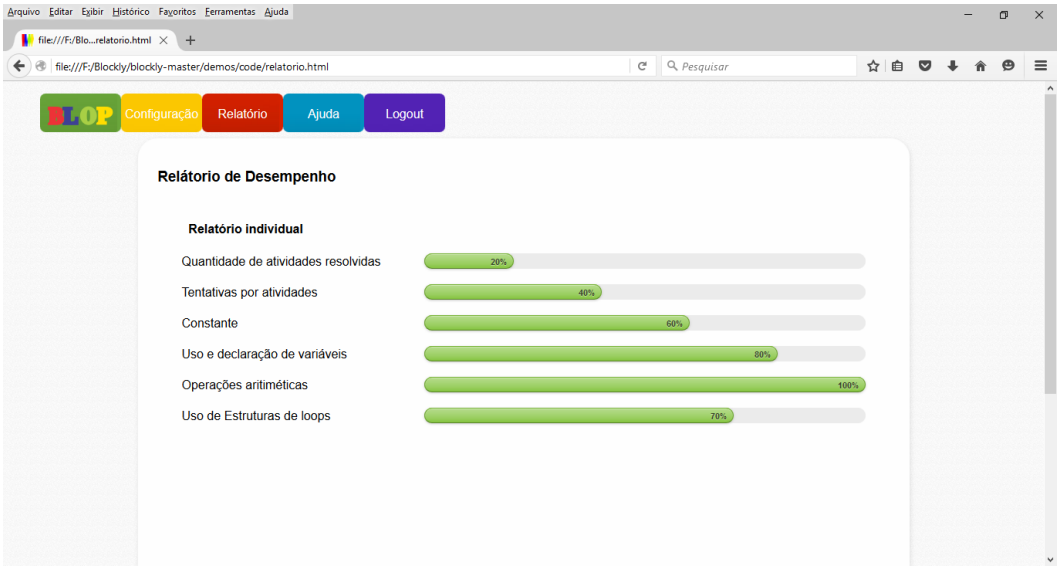


Figura 26 - Página de relatório de desempenho individual.

5.4.1 Seleção de Exercício

Uma das principais vantagens do STI BLOP é que ele orientara cada aluno para temas que são mais adequados para o seu nível atual de conhecimento. Esta orientação é feita através da lista de exercícios. O sistema exibe uma lista de exercícios que ele acha que são mais adequados para o aluno que está logado no sistema. Os exercícios são mostrados em ordem de adequação, com destaque aos exercícios mais adequado ao nível de conhecimento do aluno. Este é também o exercício que é selecionada por padrão pelo sistema logo após o aluno resolver o

questionário no primeiro acesso. O estudante pode decidir tentar outro exercício da lista, se ele desejar. A página de seleção de exercício das STI BLOP é mostrada na Figura 27.

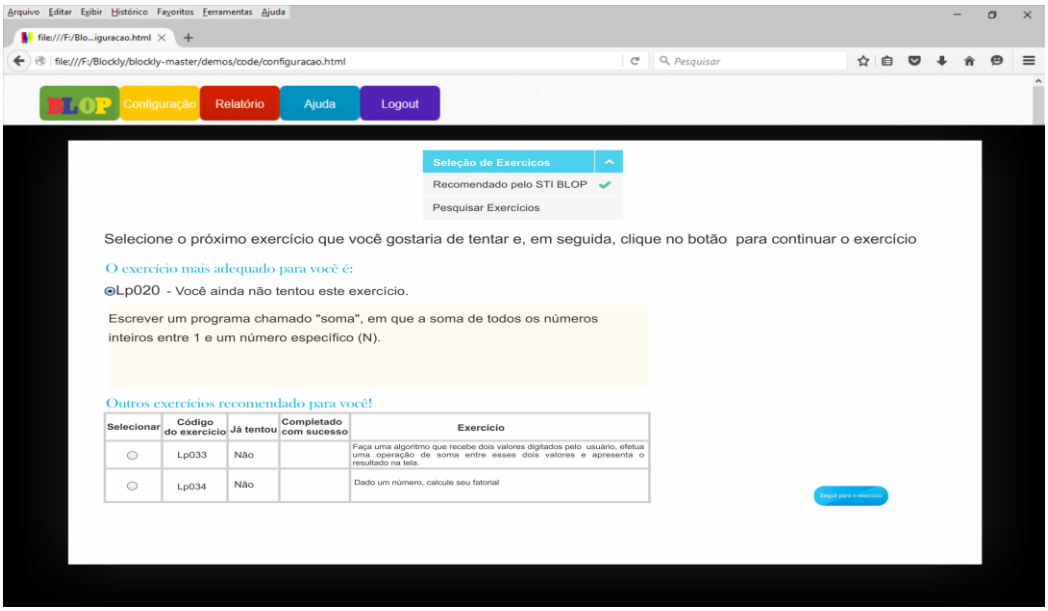


Figura 27 - Página de seleção de exercícios.

Embora o STI recomende ao aluno exercícios baseados em seu nível atual de conhecimento sobre um determinado assunto, pode haver momentos em que os alunos não concordem com as informações do sistema sobre o seu nível de conhecimento e pode querer ser encarregado de selecionar os próximos exercícios a resolver. Nesses casos, eles podem selecionar um exercício diferente na lista sugerida pelo sistema, considerando que a lista pode ser muito extensa, eles podem achar que é dificuldade em encontrar o que procuram. Uma opção de pesquisa é fornecida para este propósito. Se o estudante decide procurar um exercício, a página mostrada na Figura 28 é exibida. O aluno pode escolher qual tema precisam ser cobertos pelos exercícios que ele queira tentar resolver. Também é possível escolher exibir os exercícios que já foram tentadas, ou aqueles que ainda não foram tentados, bem como os exercícios desenvolvidos com êxito, ou os não desenvolvidos, ou ambos.

O sistema permitirá, aos alunos, a liberação de exercícios em lotes. Isso é importante porque muitos exercícios liberados, de uma única vez pode ser demais para alguns alunos. Cada exercício terá indexado uma data de lançamento.

Exercícios que tenham sido liberados, mas que tem uma data de lançamento maior do que a data especificada, são apresentados como novos exercícios. O aluno também pode selecionar a exibição de novos exercícios, os exercícios antigos ou ambos. Uma vez que escolhido os critérios de pesquisa necessários, o aluno poderá voltar à página de seleção de exercícios. Agora, esta página exibe uma lista de exercícios que correspondem aos critérios de pesquisa. O aluno pode escolher o exercício que ele quer tentar resolver.

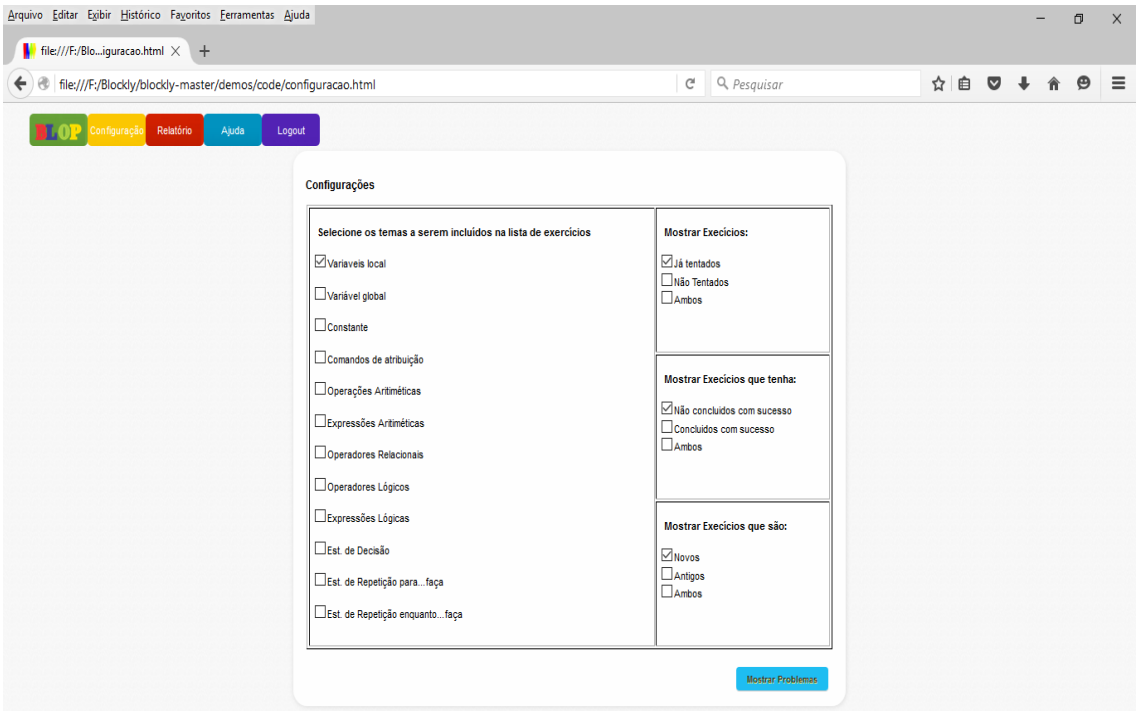


Figura 28 - Página de configuração de exercícios.

Uma vez que escolhido, os próximos exercícios devem ser sugeridos pelo sistema ou o próprio aluno deve procura-los. Está configuração, desse modo de apresentação dos exercícios, já permanecem ativas para as sessões de *login* atual, e para as sessões posteriores, a menos que o aluno mude explicitamente as configurações novamente. Isto torna mais fácil para cada aluno trabalhar com base em sua preferência de apresentação dos exercícios sem ter que escolher o modo outra veze que fizer o *login* no sistema.

Uma consideração importante neste STI está relacionada ao fato de como serão fornecidos, pelo sistema, o *feedback* ao aluno, se o *feedback* deverá ocorrer de forma proativa pelo sistema ou se o sistema deve esperar para que seja solicitado pelo aluno. Embora, muitas ideias diferentes terem sido apresentadas na literatura, este trabalho está baseado na ideia de que somente deverá ser fornecido o *feedback* ao aluno, se ele assim o deseja. Portanto, quando um aluno enviar uma solução a ser analisada pelo sistema, a única forma de *feedback*, inicialmente fornecida, será a informação que a solução está correta ou incorreta.

5.4.2 Estudo de caso (Resolvendo um Exercício)

Uma vez que um exercício é selecionado, a página para a resolução de exercício do STI BLOP será exibida. Esta página é ilustrada na Figura 29. O enunciado do exercício selecionado é exibido no topo da página. No centro da página está a área onde a resposta deve ser inserida (onde o algoritmo deverá ser construído) e no *menu* do lado esquerdo fornece todas as opções de blocos para a montagem do programa. No *menu,* na parte superior, estão disponíveis os botões para retorna a página inicial; o botão ajudar, que deve ser clicado pelo aluno afim de que o sistema lhe forneça um *feedback*; a opção de configuração do sistema, além da opção de *logout*.

A seção superior no canto direto da página estão os botões para compilar o código e o botão para salvar programas para ser adicionado a base de caso como novo problema, uma vez que não contenha quaisquer erros de lógica. Já dentro do ambiente podemos observar os *ícones* de uma lixeira que serve para descartar blocos que foram inseridos no ambiente mais que não serão usados para a construção do algoritmo, sendo assim, desnecessário, na área de montagem do programa, dois dos ícones são um sinal de “mais” e um “menos” que serve como uma lupa para dar zoom na área do ambiente de construção do algoritmo, além do ícone de centralização de página.

Enquanto monta seu programa, o sistema disponibiliza algumas opções para o aluno. Ele pode optar por salvar o programa que está montando na área do ambiente para ser usado posteriormente. O programa é então salvo uma base de dados podendo ser recarregado na área de montagem do ambiente quando o aluno

retorna ao sistema. Uma resposta por exercício pode ser salva e recarregada dessa maneira. O aluno também pode apagar tudo na área de montagem do programa e reiniciar o exercício.

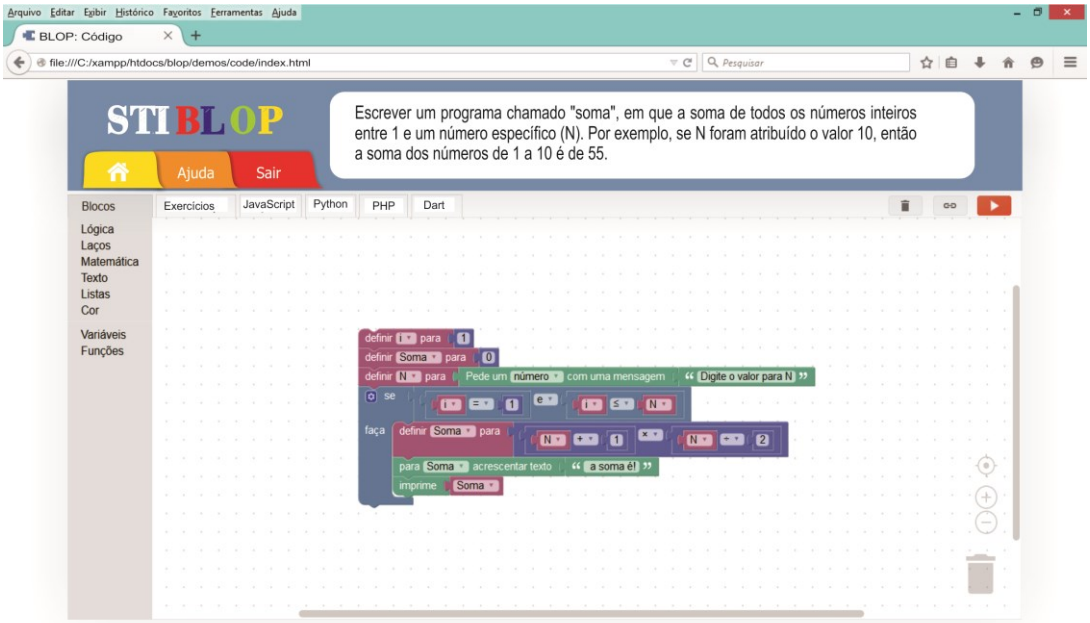


Figura 29 – Resolvendo um problema.

Assim, após o aluno montar um código ele submente o mesmo a análise do programa. Tanto a saída dos resultados do código quantos os dados para ser inseridos para processamento do programa serão exibidas e inseridos através de um tela *pop-up* que aparece para o aluno, ao ser clicado no botão de compilar programa, na parte superior direta, como já mencionado. Nessa perspectiva o código é analisado pelas STI BLOP e nenhum *feedback* é fornecido ao aluno. Portanto, o aluno é responsável por decidir se o programa está correto ou não. Para que o sistema forneça *feedback* ao aluno, é necessário que ele deva clicar no botão "ajuda". Quando este botão é usado, o sistema analisa o programa fornecido pelo aluno e inicia a fase de recuperação dos casos mais semelhantes, ao problema que está sendo resolvido, armazenados na base de casos.

O STI BLOP é um novo sistema que ainda está em desenvolvimento. Portanto, é possível que uma solução correta apresentada por um aluno esteja fora

do âmbito deste trabalho. Em tais situações, é possível que o sistema identifique uma solução correta como incorreto. Se o aluno está convencido de que a sua resposta está correta, mas o sistema se recusa a aceitá-la, ao aluno é dada a opção de registrar sua preocupação sobre este assunto e assim, um e-mail é, então, gerado ao administrador indicando a preocupação. Estes problemas podem, em seguida, ser tratado pelo administrador. Se um erro está presente no programa do aluno, uma explicação sobre o erro pode ser fornecida através de e-mail. Isto é útil, a fim de desenvolver ainda mais o sistema.

6 CONSIDERAÇÕES FINAIS E TRABALHOS FUTUROS

Ensinar programação introdutória é um grande desafio para os educadores, por muitas razões, sobre as quais apontamos em diferentes partes deste trabalho. Embora muitos métodos tenham sido sugeridos para com a finalidade de transpor este desafio, ainda se faz muito presente no cotidiano dos cursos de computação. Em particular, poucas pesquisas foram realizadas sobre os métodos de ensino de Lógica de Programação para alunos iniciantes que muitas vezes chega a essa nova área do saber com pouco ou nenhum conhecimento no assunto.

Diante desta perspectiva, uma solução para este problema, tal como sugerido pela presente investigação, é a utilização de Sistemas Tutores Inteligentes voltados para esta finalidade. O STI BLOP é um sistema que se concentra em ensinar as noções básicas de lógica de programação para alunos iniciantes no curso da área de computação. Ele fornece exercícios para os alunos com base em suas necessidades específicas, a fim de maximizar sua aprendizagem.

Para o desenvolvimento do STI BLOP está sendo realizada a integração de um sistema de raciocínio baseado em casos com um ambiente BLOP, no intuito de fornecer ao aluno uma ferramenta de resolução de problemas de algoritmo capaz de auxiliá-lo a resolver exercícios por meio de *feedback*. A utilização de ferramentas de terceiros - jColibi e o ambiente BLOP - facilitarão o desenvolvimento e possibilitarão a instanciação do sistema de raciocínio baseado em casos e do STI, de forma ágil, a fim de verificar se a solução proposta é capaz de atingir os objetivos deste trabalho. Diante do exposto, acreditamos ser possível analisar as formas adequadas de representar e recuperar o caso no domínio de auxílio ao ensino de lógica de programação e o *feedback* será adequado para estimular a habilidade dos alunos em identificar similaridades entre os problemas e a partir daí construir seu raciocínio para o desenvolvimento e resolução dos problemas.

Foi descrito nesse trabalho um modelo computacional de um Sistema Tutor Inteligente chamado STI BLOP. O STI BLOP que está sendo projetado para o ensino no domínio de Lógica de Programação com a finalidade de ajudar alunos a superar suas dificuldades na apropriação de tal conhecimento. O BLOP, além de possibilitar aos alunos apropriação de conhecimento no domínio de Lógica de

Programação, tem a pretensão de administra automaticamente problemas, gerando dinamicamente e adaptando *feedback* em tempo de execução de forma individualmente a cada aluno.

Conforme descrito nas seções anteriores, muitas representações de conhecimento têm sido utilizadas por pesquisadores a fim de representar a base de conhecimento em Sistemas de Tutores Inteligentes projetados para ensinar programação. Cada uma dessas representações tem muitas vantagens, mas também algumas deficiências. Portanto, uma abordagem totalmente nova foi utilizado durante o projeto do STI BLOP. A principal exigência da representação foi que ele seja capaz de suportar o raciocínio lógico sobre a estrutura de programas escritos por alunos e fornecer *feedback* adequado com base nos erros específicos. Técnicas de inteligência artificial pareciam ser um meio muito razoável de atingir este objetivo. Dos muitos formalismos disponíveis em AI, o Raciocínio Baseado em Casos é uma representação bastante simples, mas poderosa. Portanto, parecia um bom candidato a ser usado para esta finalidade, embora nenhum trabalho anterior parece ter olhado para essa possibilidade.

A linguagem de programação visual usada aqui, se apresenta como um recurso a ser utilizado como facilitadora no processo de ensino e aprendizagem de programação de computadores na disciplina de Lógica de Programação para iniciantes. Uma avalição inicial do BLOP será realizada após o protótipo ser totalmente desenvolvido, essa avaliação será realizada juntamente com docentes e discentes do curso de Computação de uma Universidade, no Brasil. Assim pretendemos realizar uma avaliação mais abrangente do sistema afim de fornecer um ambiente de aprendizagem interativo e rico para os alunos o que resultará em um aumento significativo de suas conquistas.

A linguagem visual é uma linguagem que é usada, atualmente, em muitos jogos para o ensino de princípios básicos de programação a crianças. As possíveis estruturas na linguagem visual são numerosas e era praticamente impossível para lidar com todos esses elementos durante as limitações de tempo de uma dissertação. Portanto, apenas um subconjunto da língua foi considerado adequado para ensinar um programador iniciante. Isto significa que tópicos mais avançados da linguagem de programação visual foram ignorados.

Os diferentes sistemas que trabalham com inteligência artificial são dependentes do domínio no qual estão inseridos. Uma das maiores dificuldades em desenvolver um STI é modelar os dados de forma que a aplicação possa utilizá-los. No entanto, no desenvolvimento do sistema podemos inserir outras funcionalidades que podem vir a melhoradas o ambiente, que incluem:

- Criação de novas medidas de similaridade: Como as medidas de similaridade são importantes para o bom funcionamento de um sistema RBC e por serem dependentes do domínio, o ambiente poderia oferecer novas medidas de similaridade;
- Definição de medidas de similaridade local: As medidas de similaridade proposta no ambiente consideram apenas cálculos globais e locais dos pesos dos atributos. Métricas de similaridade por *string* podem ser utilizadas para melhorar a recuperação de casos semelhantes;
- Melhora o processo de inclusão de peso aos casos: Além da descrição dos atributos pertencentes aos casos, poderia ser incorporado pesos aos mesmos, de forma que possa ser utilizado para mensurar o grau de conhecimento do usuário que o realizou;
- Estruturação do conteúdo programático: A organização da base de conhecimento do ambiente se prendeu apenas em questões práticas e questionários, ou seja, todo o conteúdo programático de ensino do STI não foi considerado. Seria interessante otimizar o ambiente tentando oferecer uma forma de organizar esse conhecimento também, facilitando a estruturação de todo a aplicação;
- Associação de tópicos aos casos: Um caso representa um episódio já ocorrido. Através desse episódio é possível identificar habilidades e deficiências do usuário que o experimentou. Através dessa identificação, os casos poderiam ser associados a tópicos que poderiam ser sugeridos para leitura e estudo por parte do aluno.

7 REFERÊNCIAS

ABEL, M. **Um estudo sobre o Raciocínio Baseado em Casos**. Porto Alegre: Universidade Federal do Rio Grande do Sul, 1996. Disponível em: http://www.inf.ufrgs.br/bdi/wp-content/uploads/CBR-TI60.pdf. Acesso em:

ALMEIDA, E. S. *et al.*. AMBAP: **Um Ambiente de Apoio ao Aprendizado de Programação**. Disponível em: <http://www.lbd.dcc.ufmg.br/colecoes/wei/2002/006.pdf>. Acesso em: 02 mar. De 2015.

BEZERRA, E. **Princípios de análise e projeto de sistemas com UML**. 2. ed. Rio de Janeiro: Editora Campus, 2006.

BOTELHO, C. A.. **Sistemas Tutores no domínio da programação**. Revista Informática Aplicada v IV nº1 jan/jun 2008. Disponível em: http://www.ria.net.br/index.php/ria/article/view/34/34. Acesso em: 03 de out. 2015.

CARVALHO, S. D. de. **Modelo híbrido de sistema tutor inteligente utilizando conhecimento do especialista e mapas de Kohonen com treinamento automatizado**. 2012. 227 f. Tese (Doutorado em Engenharia Elétrica). Faculdade de Engenharia Elétrica da Universidade Federal de Uberlândia. Uberlândia. Disponível em: http://biblioteca.versila.com/14561015. Acesso em: 05 de out. 2015.

CORMEN, T. H. et. al. **Algoritmos**: teoria e prática. 2.ed. Rio de Janeiro: Elsevier:2002.

FORBELLONE, A. L V.; EBRSPÄCHER, H. F. **Lógica de Programação – A Construção de Algoritmos e Estruturas de Dados**. São Paulo: Makron Books, 1993.

GOMES, A., & MENDES, A. J.. **Learning to program - difficulties and solutions**. International Conference on Engineering Education – ICEE. 2007. Disponível em: < http://www.ineer.org/Events/ICEE2007/papers/411.pdf>. Acesso em: 30 jul. 2015.

HOSTINS, H. RAABE, A. **Auxiliando a Aprendizagem de Algoritmos com a Ferramenta WEBPORTUGOL**. XV Workshop sobre Educação em Computação. Rio

de Janeiro. Anais do XXVII Congresso da SBC. p. 96 – 105. 2007. Disponível em: <http://www.ldb.dcc.ufmg.br/colecoes/wei/2007/0010.pdf>. Acesso em: 05 sent. 2015.

IEPSEN, E F.. **Ensino de algoritmos: detecção do estado afetivo de frustração para apoio ao processo de aprendizagem**. 2013. 157 f. Tese (Doutorado em Informática na Educação). Universidade Federal do Rio Grande do Sul - UFRGS. Rio Grande do Sul. Disponível em http://www.lume.ufrgs.br/bitstream/handle/10183/78020/000898031.pdf?sequence=1. Acesso em: 02 de nov. 2015.

SILVA. J. R. DA. **Aplicação da Técnica RBC no Sistema de Informação para Suporte aos Clientes da Empresa Multitherm Sistemas e Automação Ltda**. 2011. Conclusão de Curso submetido (Monografia). Universidade Regional de Blumenau. Blumenau. Disponivel em: http://dsc.inf.furb.br/arquivos/tccs/monografias/TCC2011-1-12-VF-JoaoRSilva.pdf. Acesso em:

JOHNSON, W. L. (1990). **Understanding and debugging novice programs**. Artificial Intelligence. Journal Artificial Intelligence - Special issue on artificial intelligence and learning environments archive Volume 42, Issue 1, Feb. 1990. Pages 51 – 97.

GRAVIDIA, J. J. Z.; ANDRADE, L. C. V. de. **Sistemas Tutores Inteligentes**. 2003. Rio de Janeiro. Disponível em: http://cos.ufrj.br/~ines/courses/cos740/leila/cos740/STImono.pdf. Acesso em: 23 set. 2015.

MAIN, T. D. J.; S. SHIU. **A Tutorial on Case-Based Reasoning**. Soft Computing in Case Based Reasoning. Springer-Verlag (London) Ltd, 2001. pp.1-28. Disponível em: http://www4.comp.polyu.edu.hk/~csckshiu/pdf/shiu01scbrb2.pdf. Acesso em: 05 out. 2015.

MARCONI, M. A.; LAKATOS, E. M. **Técnicas de Pesquisa**. Editora Atlas, São Paulo, SP, 1996.

MENDES, A. J. N. ***Software* educativo para apoio à aprendizagem de programação**. In: Taller International de *Software* Educativo, Santiago - Chile, 2002. Disponível em <http://www.c5.cl/ieinvestiga/actas/tise01/pags/charlas/charla_mendes.htm>.

MENDES, J. B. **Um framework de raciocínio baseado em casos aplicados para estruturar a base de conhecimento em sistemas tutores inteligentes**. 2012. 147 f. Dissertação (Mestrado em Ciências e Tecnologia da Computação). Programa de Pós-Graduação em Ciência e Tecnologia da Computação, Universidade Federal de Itajaí. Itajubá (MG).

MITROVIC, A; OHLSSON , S . **Evaluation of a Constraint Based Tutor for a Database Language**. International Journal of Artificial Intelligence in Education. V. 10. p. 238-256. 1999. Disponível em: < http://www.iaied.org/pub/987/file/987_paper.pdf>. Acesso em: 17 jul. 2015.

MORTARI, C. A. **Introdução à Lógica**. Editora UNESP: Imprensa Oficial do Estado, 2001. Bibliografia. ISBN 85-7139-337-0 (Editora UNESP) 85-7060-182-4 (Imprensa Oficial do Estado) 1. Lógica 2. Filosofia 1. Titulo. li. Série.

NASER, S. S. A.. **Developing an Intelligent Tutoring System For Students Learning To Program in C++**. Information Technology Journal. 2008. Disponível em: http://docsdrive.com/pdfs/ansinet/itj/2008/1055-1060.pdf. Acesso em 04 de nov. 2015.

NOBRE, I. A. M. N., MENEZES, C. S. **Suporte à Cooperação em um Ambiente de Aprendizagem para Programação**. XIII Simpósio Brasileiro de Informática na Educação – SBIE 2002. São Leopoldo, 2002.

OLIVEIRA, M. K. de. **Aprendizagem e Desenvolvimento**: um processo sócio histórico. São Paulo: Scipione, 2009.

PETRY, P. G.. **Um sistema para o ensino e aprendizagem de algoritmos utilizando um companheiro de aprendizagem colaborativo**. 2008. 86p. Tese (Mestrado em Ciência da Computação), Universidade Federal de Santa Catarina. Florianópolis, 2005.

PILLAY, N. (2003). **Developing intelligent programming tutors for novice programmers**. Disponível em: <http://delivery.acm.org/10.1145/790000/782986/p78pillay.pdf?ip=200.137.128.130&id=782986&acc=ACTIVE%20SERVICE&key=344E43C9DC262BB%2ECE27CB3E373CC636%2E4D4702B0C3E38B35%2E4D4702BC3E38B35&CFID=549988525&CFTOKEN=48169253&__acm__=1443713664_800216ce66da053f50be3002833e38a>. Acesso em: 10 jun. 2015.

PIMENTEL, E. P.; FRANÇA, V. F.; OMAR, N. **A Caminho de um Ambiente de Avaliação e Acompanhamento Contínuo da Aprendizagem em Programação de Computadores**. II Workshop de Educação em Computação e Informática do Estado de Minas Gerais, 2003. Acessado em 29-12-2004.

PRUS, É. M.. **Um modelo de um sistema tutor inteligente aplicado ao ensino da programação estruturada**. 2001. 131 f. Dissertação (Mestrado em Engenharia de Produção). Pós-Graduação em Engenharia da Produção da Universidade Federal de Santa Catarina. Santa Catarina - SC Disponível em: https://repositorio.ufsc.br/handle/123456789/81900. Acesso em: 07 de nov. 2015.

RISCO, S., & REYE, J. (2009). **Personal Access Tutor - helping students to learn MS Access**. In V. Dimitrova, R. Mizoguchi, B. Du Boulay & A. C. Graesser (Eds.), 14th International Conference on Artificial Intelligence in Education. Vol. 200. p. 541-548. 2009. Brighton.

RODRIGUES JUNIOR, M. C.. **Experiências Positivas para o Ensino de Algoritmos**. Disponível em: http://www.uefs.br/erbase2004/documentos/weibase/Weibase2004Artigo001.pdf. Acesso em: 05 abr. 2015.

SANTOS, R.P.*et al.*. **Uma Proposta de Cenário para Ensino de Algoritmos e Programação com Contribuições de Cooperação, Colaboração e Coordenação**. 2008. In: Anais do XVI Workshop sobre Educação em Computação, XXVIII Congresso da Sociedade Brasileira de Computação, Belém, PA, Brasil, 218-227.

SELF, J.. **Bypassing the intractable problem of student modelling**. In C. Frasson & G. Gauthier (Eds.), Intelligent tutoring systems: At the crossroads of artificial intelligence and education. 1990. p. 107-123. Norwood, N.J, USA: Ablex. Disponível

em: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.55.2809&rep=rep1&type=df. Acesso em: 08 de dez. 2015.

SIEBRA, Sandra de Albuquerque; Silva, Danielle Rousy D. da. **Prática de Ensino de Algoritmos**. Disponível em: <http://pt.scribd.com/doc/27483736/Pratica-de-Ensino de Algoritmo-volume-1e2#scribd>. Acesso em: 20 abr. 2015.

SOMMERVILLE, I. **Engenharia de *software***. 8. ed. São Paulo: Addison Wesley, 2007.

SYKES. E. R.. **An Intelligent Tutoring System Prototype for Learning to Program JavaTM**. School of Computing and Information Management, Sheridan College 1430 Trafalgar Road, Oakville, Ont., Canada, L6H 2L1 +1 (905) 845-9430 Ext. 2490 ed.sykes@sheridanc.on.ca . Franya Franek Department of Computing and *Software*, Faculty of Science, McMaster University 1280 Main Street W., Hamilton, Ont., Canada.

TRUONG, N., Bancroft, P., & Roe, P.. **A *web* based environment for learning to program**. Proceedings. 26th Australasian Computer Science Conference. V. 16. 2003. p. 255-264. Disponível em: http://crpit.com/confpapers/CRPITV16Truong.pdf. Acesso em: 05 de dez. 2015.

URNAU, E; KIPPER, L. M; FROZZA, R. **Desenvolvimento de um sistema de apoio à decisão com a técnica de raciocínio baseado em casos**. Perspect. ciênc. inf. [online]. 2014, vol.19, n.4, p. 118-135. ISSN 1413-9936. Disponível em: http://portaldeperiodicos.eci.ufmg.br/index.php/pci/article/view/1636/1511. Acesso em: 06 de out. de 2015.

VALENTIN, H. K. A. **Um Estudo sobre o Ensino-aprendizagem de Lógica de Programação**. Encontro Nacional de Pesquisa em Educação e Ciência. 2009. Disponível em: <http://posgrad.fae.ufmg.br/posgrad/viienpec/pdfs/137.pdf>. Acesso em: 05 abr. 2015.

VANLEHN, K. S. S.; Murray, C.; Baggett, W. **What makes a tutorial event effective**. In M. A. Gernsbacher & S. Derry (Eds.), Proceedings of the Twenty-First Annual

Conference of the Cognitive Science Society. 1998. Hillsdale, NJ, USA: Erlbaum. Disponível em: citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.115.1180&rep=rep1&type=pdf.

VYGOTSKY, L. S.. **A Formação Social da Mente**. 4.ed. São Paulo: Martins Fontes: 1991.

WANGENHEIM, C. G.; WANGENHEIM, A. V.; RATEKE T. **Raciocínio Baseado em Casos**. 2º. ed. [S.l.]: Manole, 2013.

WATSON, I. & MARIR, F. **Case-Based Reasoning**: A Review, The Knowledge Engineering Review 9(4), pp. 355-381, 1994.

WENGER, E. **Artificial Intelligence and Tutoring Systems**: Computational and Cognitive Approaches to the Communication of Knowledge. Morgan Kaufmann. 1987. San Francisco, CA, USA. ISBN:0-934613-26-5.

WOOLF, B. P.. **Building Intelligent Interactive Tutors: student-centered strategies for revolutionizing e-learning**. Burlington, MA: Morgan Kaufman. 2009. ISBN: 978-0-12-373594-2.

APÊNDICE: ARTIGOS PUBLICADOS

Neste apêndice são apresentados os trabalhos publicados sobre o tema, que indicam as etapas elaboradas durante a realização desse trabalho de dissertação.

A.1. **Proposta de um Ambiente Online para a Resolução e Correção Automática de Algoritmo**

Autores: Idovaldo Cunha da Silva, Luis Carlos Costa Fonseca, Reinaldo de Jesus da Silva

Evento: XX Congresso Internacional de Informática Educativa – TISE

Local: Santiago - Chile

Categoria: Short Paper

Ano: 2015

Resumo: Com o intuito de contribuir no processo de ensino e aprendizagem na disciplina de lógica de programação, este trabalho visa desenvolver uma ferramenta de autoria e aplicação de prova com correção automática. Para tanto, integraremos a ferramenta a um editor visual de código ao ambiente BOCA, que é um sistema utilizado para gerenciar competições de programação de computadores. O desenvolvimento da ferramenta busca propiciar meios fundamentados pedagogicamente nos pressupostos teórico-metodológicos da Teoria Histórico-Cultural, para a realização de provas de lógica de programação em que as respostas sejam algoritmos que serão corrigidos automaticamente, gerando um retorno imediato para o aluno.

A.2. **Um Sistema Tutor Inteligente para o Ensino no Domínio de Lógica de Programação**

Autores: Idovaldo Cunha da Silva, Luis Carlos Costa Fonseca, Reinaldo de Jesus Silva

Evento: XX Congresso Internacional de Informática Educativa – TISE

Local: Santiago - Chile

Categoria: Short Paper

Ano: 2015

Resumo: Este trabalho objetiva apresentar uma modelo computacional de um Sistema Tutor Inteligente (STI), para suporte ao processo de ensino e aprendizagem no domínio de lógica de programação. O STI utiliza como técnica de Inteligência Artificial - IA, o Raciocínio Baseado em Casos (Case-based Reasoning - CBR), fazendo uso de uma base de conhecimento do modelo do domínio. Assim, este artigo aborda a concepção, modelagem e prototipação de um Sistema Tutor Inteligente para ensinar lógica de programação fazendo uso de uma linguagem visual. A investigação está em andamento e partir da hipótese de que o protótipo concluído será suficiente para provar o conceito de que o STI BLOP, totalmente desenvolvido, irá fornecer um ambiente de aprendizagem interativo e rico para os alunos o que resultará em um aumento significativo de suas conquistas.

A.2. **Um Modelo de Sistema Tutor Inteligente para o Ensino no Domínio de Lógica de Programação**

Autores: Idovaldo Cunha da Silva, Luis Carlos Costa Fonseca, Reinaldo de Jesus da Silva, Angelo Rodrigo Bianchini

Evento: COMPUTER ON THE BEACH

Local: Florianópolis - Brasil

Categoria: Full Paper

Ano: 2016

Resumo: Este artigo apresentar uma modelo computacional de um Sistema Tutor Inteligente (STI), para suporte ao ensino no domínio de lógica de programação. O

STI utiliza como técnica de Inteligência Artificial - IA, o Raciocínio Baseado em Casos (RBC), fazendo uso de uma base de conhecimento do modelo de domínio. Assim, este artigo aborda a concepção, modelagem e prototipação de um STI para ensinar lógica de programação fazendo uso de uma linguagem visual. A investigação em andamento e partir da hipótese de que o protótipo concluído será suficiente para provar o conceito de que o STI BLOP, totalmente desenvolvido, irá fornecer um ambiente de aprendizagem interativo e rico para os alunos.