Uma Arquitetura de Tutor Inteligente que Provê Suporte ao Diálogo com o Aluno Iniciante em Linguagem de Programação

Renato de M. Santos[1], Crediné Silva de Menezes [1], Davidson Cury [1]

[1]Programa de Pós-Graduação em Informática - Universidade Federal do Espírito Santo (UFES) - Caixa Postal l01.9011 - CEP 29.075-910 – Vitória – ES – Brasil

{renatosantos.ti, credine, dedecury}@gmail.com

Abstract. *This article presents an architecture of an Intelligent Tutor System (ITS) that supports the implementation of dialog resources in a more contextualized way and supporting the needs registered in the student model during the introductory learning of programming language using Python. The proposed ITS improves the teacher's work and extends its attendance capacity.*

Resumo. *Este artigo apresenta uma arquitetura de um Sistema Tutor Inteligente (STI) que suporta a implementação de recursos de diálogo de forma mais contextualiza e atendendo às deficiências registradas no modelo do aluno durante o aprendizado introdutório de linguagem de programação usando o Python. O STI proposto busca melhorar o trabalho do professor e estende sua capacidade de atendimento.*

1. Introdução

O uso dos sistemas tutores inteligentes (STI) no ensino de programação é uma alternativa para potencializar o trabalho do professor e aumentar a eficácia do aprendizado durante a realização das disciplinas iniciais de programação, pois permite reduzir ou eliminar deficiências encontradas no percurso, tais como a falta de um ensino individualizado, a necessidade de um entendimento rápido do perfil e dos anseios do aluno bem como a ausência de acompanhamento fora do ambiente de aula.

A conversação é uma importante funcionalidade empregada em tutores inteligentes, para [Tobar et al. 2001], uma interface em língua natural motiva a participação dos aprendizes e desempenha um papel importante no aprendizado. Esse tipo de interação permite que os alunos se relacionem de forma mais humana com o computador [Coronado et al. 2018]. No entanto, segundo [Medeiros, Moser e Dos Santos 2013], essa funcionalidade é normalmente baseada na análise de perguntas digitadas pelo aluno e os sistemas são apenas reativos. Tutores inteligentes que possam captar diversas informações sobre o contexto e sobre o aluno no processo de construção do diálogo ainda é um problema identificado nos tutores voltados ao ensino de programação.

Esse trabalho concentra-se na definição de uma arquitetura de STI que subsidie o módulo de tutoria no tratamento do diálogo com o aluno iniciante em programação, levando em consideração suas necessidades e as informações obtidas do contexto durante esse diálogo. Propõe-se modificações na arquitetura tradicional de um STI,

DOI: 10.5753/cbie.wcbie.2018.768

incorporando, dentre outros aspectos, um modelo do aluno que registra diferentes tipos de conhecimento sobre programação, um módulo de tutoria que avalia as possíveis intenções e necessidades desse aluno ao realizar uma pergunta.

A Seção 2 aborda as metodologias utilizadas no ensino de programação e as dificuldades encontradas neste processo. Na Seção 3 serão apresentados os trabalhos correlatos, que empregam o uso de tutores inteligentes para o ensino de programação, e como o dialogo tem sido tratado nesses tutores. A Seção 4 expõe a caracterização do problema de pesquisa e a metodologia empregada neste trabalho. A Seção 5 apresenta a arquitetura proposta, resultado dessa pesquisa. Na Seção 6 serão abordadas as possíveis técnicas para a implementação da arquitetura proposta. Por fim, na Seção 7 serão apresentadas as conclusões os trabalhos futuros vislumbrados durante esse estudo.

2. O Ensino e Aprendizagem de Programação

As dificuldades encontradas no ensino introdutório de programação podem ser diagnosticadas pelo alto grau de repetência e pelas dificuldades dos alunos em disciplinas avançadas que exigem como pré-requisito o domínio da criação de algoritmos [Tobar et al. 2001], [Júnior, Fechine e Costa 2009] aponta, dentre outras, dificuldade de o aluno combinar os artefatos aprendidos para a resolução de novos problemas, enquanto que [Moreira et al. 2018] ao investigar 110 alunos, registra que as dificuldades essas ainda persistem e são preocupantes. Suas conclusões indicam que a maior dificuldade desses alunos está no desenvolvimento da lógica de programação (42.72%), no entendimento da sintaxe (34.54%), na falta de tempo para se dedicar a disciplina (26.36%) e a dificuldade na interpretação das questões (15.45%).

Por outro lado, [Aureliano, Tedesco e Giraffa 2016] argumenta que é extremamente difícil para o professor conhecer o perfil de cada aluno, seus objetivos, suas dúvidas e as suas condições para apreender um novo conhecimento, logo o professor torna-se impotente ao tentar proporcionar a cada aprendiz um ensino adaptado às suas necessidades e expectativas [Falckembach e Araujo 2013].

2.1 Metodologias no Ensino de Programação

A POO é defendida por [Kölling 2003] pelo seu alto grau de reusabilidade e sua fácil assimilação dos objetos criados na linguagem com àqueles do mundo real. Por outro lado, [Falckembach e Araujo 2013] argumentam que o ensino de algoritmos é mais eficiente com o uso da programação estruturada, pois, facilita ao aluno compreender e explicitar as ações que compõem um programa. [Koliver, Dorneles e Casa 2004] defendem ambientes livres de paradigmas e de linguagens como uma forma de abstrair detalhes desnecessários, pois tal abordagem direciona o raciocínio para resolução do problema. Nessa direção, há ferramentas visuais tais como o *App Inventor* e *Scratch*, que utilizam formas gráficas para representar os comandos e as estruturas da linguagem [Bombasar et al. 2015].

2.2 Tutores Inteligentes Aplicados ao Ensino de Programação

A pesquisa em torno de STI voltado para o ensino de programação tem gerado diversas linhas de investigação, tais como a identificação de erros cometidos pelos alunos durante a resolução do problema, o rastreamento das estratégias do aluno durante a

resolução das atividades e a correção de conceitos malformados durante o aprendizado [Ueno 1989].

Inúmeras outras características têm sido implementadas nos tutores inteligentes, por exemplo: a disponibilidade desses tutores na web e em dispositivos móveis, os estilos de aprendizagem, o tratamento de emoções e a aplicação de variadas abordagens pedagógicas. Trabalhos recentes como [El Haddad e Naser 2017] e [Price, Dong e Lipovac 2017], apontam que os atuais tutores inteligentes ainda falham em medir a eficácia e os impactos de suas sugestões fornecidas, pois são deficientes em oferecer *feedback* imediato e não são eficientes ao questionar os alunos para que estes possam esclarecer suas ações.

3. Trabalhos Correlatos

[Ueno 1989] propôs o *INTELLITUTOR*, um STI que realizava a identificação de erros e inferia a estratégia do aluno através do *Hierarchical Procedure Graphic* (HPC), assim fazia sugestões e alertava sobre os erros cometidos. Semelhantemente, [Gerdes et al. 2017] desenvolveram o *Ask-Elle*, um STI para ensino da linguagem funcional *Haskell*. O *Ask-Elle* rastreia as estratégias do aluno na solução do problema, e para o fornecimento de dicas ele realiza diversas derivações de uma solução modelo do problema. No trabalho de [Price, Dong e Lipovac 2017] o STI *iSnap*, rastreia o estágio do código do aluno e fornece dicas em formato de blocos de códigos prontos com a finalidade de orientar o aluno a melhorar seu programa ou resolver uma dificuldade. Por se limitar à análise do código, o *iSnap* ignora o modelo do aluno e outras informações de contexto.

Para [Coronado et al. 2018] e [Tobar et al. 2001] a aplicação da linguagem natural facilita a interação dos alunos com os tutores inteligentes, sendo que os avanços na computação cognitiva permitem um novo modo de interação que acelera a percepção das variadas fontes de informações existentes. Muitos tutores inteligentes que tratam a conversação têm feito uso da *AIML* (*Artificial Intelligence Markup Language),* uma linguagem de marcação baseada em *XML*, todavia, segundo [Paz et al. 2017], por ser um sistema baseado em regras, a *AIML* torna complexo e dispendioso o tratamento de respostas no diálogo. O trabalho de [Júnior, Fechine e Costa 2009] empregou *AIML* num STI para ensino de Python, porém apresenta lacunas por empregar um sistema puramente baseado em regras e por não considerar o modelo do aluno.

O *iAIML*, foi uma alternativa proposta por [Das Neves 2005] com o objetivo de fornecer um tratamento de intenções na linguagem *AIML*, para isto utilizou-se da Teoria da Análise da Conversação (TAC). Todavia, esse trabalho limitou o tratamento do diálogo apenas sob a perspectiva da avaliação do texto envolvido na pergunta, ignorando o perfil e o momento do aluno durante a dúvida. Complementarmente, [Medeiros, Junior e Moser 2017], apresentam o *THOTH*, capaz de enunciar *small talks,* que são pequenas conversações cujo propósito é manter um canal de comunicação aberto entre os interlocutores. Muito embora esse trabalho tenha destacado a importância de um diálogo mais amigável como uma característica valiosa num STI, não foi implementado no *THOTH* um diálogo tutorial condizente com as necessidades do aluno.

4. Caracterização do Problema

Identificar o que aluno está perguntado e descobrir uma resposta adequada, levando em consideração as condições deste aluno para a recepção da resposta, bem como agregar o contexto da pergunta, ainda é um problema atual nos sistemas tutores inteligentes. Esse problema ainda é mais agravante quando considerado o âmbito de STI´s voltados ao ensino de lógica programação. Por exemplo, dada uma certa pergunta, como identificar se um aluno não sabe nada sobre o assunto e necessita de uma resposta abrangente ou se apenas demonstrar um exemplo do uso de certo comando atenderá a necessidade desse aluno? Para [Allen et al. 2001], essa dificuldade está ligada aos limites das teorias utilizadas na construção de um STI, visto que muitos desses sistemas não tratam a intenção vinculada ao contexto em que a mensagem está inserida.

O *iAIML* foi uma alternativa mais dinâmica para o reconhecimento de intenções, porém até o presente trabalho, não há registros da aplicação do *iAML* no contexto de tutores inteligentes para o ensino de programação. Em [De Oliveira et al. 2010], verifica-se o emprego do *iAML* na construção de um STI, no entanto, a arquitetura implementada apresenta a ausência do modelo do aluno, impedindo que esse tutor agregue as informações sobre o conhecimento e deficiências do aprendiz durante o diálogo.

Delimitou-se o escopo deste trabalho ao ensino de algoritmos e de lógica de programação a alunos iniciantes. A arquitetura proposta busca fornecer um atendimento às dúvidas do aprendiz antes e durante a realização das atividades, o suporte na utilização da sintaxe e da semântica da linguagem; o fornecimento de exemplos no uso de comandos, a explicação das causas de erros e a obtenção de respostas para as dúvidas esperadas. A linguagem utilizada será o Python, muito embora a arquitetura possa ser portada para outras linguagens.

5. Arquitetura Proposta

A Figura 1 ilustra nos retângulos externos a arquitetura tradicional de um STI e internamente nesses módulos são apresentadas as modificações propostas nesse trabalho. O ***Modelo do Aluno*** foi subdividido em sub módulos, o objetivo é registrar o conhecimento conceitual sobre os elementos da linguagem (C1), o domínio da sintaxe/semântica (C2) e a capacidade deste aluno escolher usar determinados recursos quando esses são necessários na resolução do problema (C3).

No módulo ***Tutor*** o sub módulo ***Analisador de Intenções*** será responsável por identificar as intenções prováveis com base na análise textual, enquanto que o ***Gerenciador de Diálogo*** tratará essas intenções com base no ***Modelo do Aluno*** e nas diversas informações de contexto fornecidas pela interface com o usuário.

Propõe-se que o STI seja disponibilizado através de uma interface web, permitindo que o aluno acesse o ambiente a qualquer momento e de qualquer local. O módulo ***Repositório de Problemas*** é composto por atividades criadas pelo professor e acessível ao aluno conforme período definido. Na ***Caixa de Diálogo*** o aluno poderá realizar perguntas e tirar suas dúvidas a qualquer tempo, inclusive no contexto de resolução de alguma atividade.

Para experimentos e testes rápidos de comandos será disponibilizado o módulo ***Caixa de Comandos*** que possibilitará que o aluno exercite a sintaxe e a semântica da linguagem Python ou realize micro tarefas sugeridas pelo STI durante o diálogo ou na identificação de alguma inabilidade desde aluno.

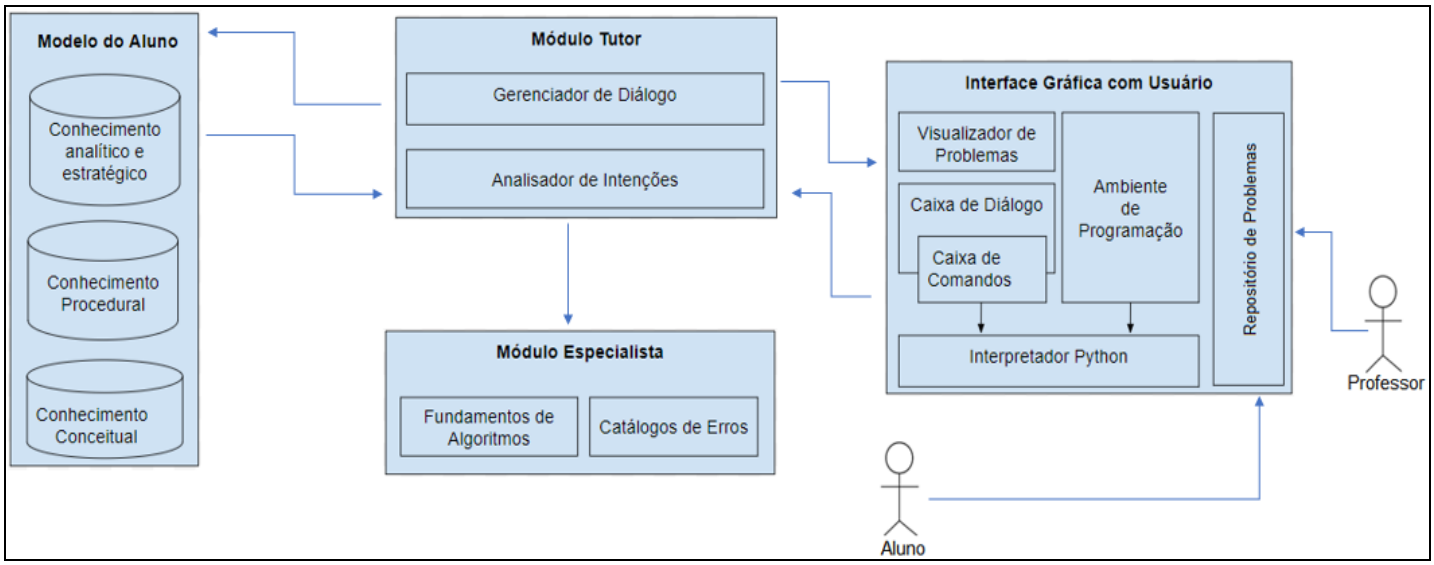

Figura 1. Arquitetura do STI proposto

5.1. Modelo do Aluno

A concentração de esforços no aluno é uma das características fundamentais num STI e o ***Modelo Aluno*** é responsável por garantir o ensino personalizado e devidamente adaptado às necessidades individuais desse aprendiz [Vier, Gluz e Jaques 2015].

Propõe-se neste trabalho que o modelo do aluno registre o conhecimento do aluno sob três aspectos: o ***Conhecimento Conceitual*** (C1) estar relacionado ao saber terminológico de cada elemento e estrutura ligada a um algoritmo; O ***Conhecimento Procedural*** (C2) representa a capacidade de o aluno utilizar corretamente os elementos e estruturas da linguagem, aplicando-os corretamente em termos de sintaxe e semântica e o ***Conhecimento Estratégico e Analítico*** (C3) que indica a habilidade do aprendiz escolher e combinar adequadamente elementos da linguagem ao solucionar determinado problema, mesmo quando estes elementos não explicitados no enunciado.

5.2. Módulo Tutor

Tradicionalmente o módulo tutor possui um papel de intercambiar a comunicação com o aluno e é efetivamente responsável por realizar a tutoria. Neste trabalho foi proposto a modificação desse módulo, dividindo-o em dois sub módulos: o ***Analisador de Intenções*** e o ***Gerenciador de Diálogo***. Tal modificação objetiva separar a responsabilidade de tratamento de linguagem natural, identificando as possíveis intenções com base nos textos e nos turnos na conversação, enquanto que o módulo Gerenciador de Diálogo ficará dedicado a avaliação dessas intenções frente ao contexto e seleção de uma resposta condizente com a intenção identificada e a estado atual do aluno.

O Gerenciador de Diálogo será responsável pela escolha da intenção usada para o fornecimento da resposta. A Figura 2 ilustra um caso hipotético de um ***Aluno A*** que apresenta uma pergunta sobre o uso dos operadores condicionais com o ***IF***.

Passo 1: O aluno ***A*** realiza a pergunta, ***como testar se o valor de uma variável é maior que 10***?
Passo 2: Identifica-se no texto as intenções prováveis e suas respectivas probabilidades;

Passo 3: As intenções prováveis são analisadas sob a perspectiva contextual através da consulta às dicas registradas na atividade, às respostas aos erros esperados e aos problemas ocorridos no código atual;

Passo 4: O Modelo do Aluno é analisado e é identificada a condição do aluno em cada assunto relacionado às intenções prováveis obtidas no Passo 2.

Passo 5: A partir das diversas informações obtidas sobre cada intenção, o módulo Gerenciador de Diálogo infere e decide por qual intenção será realizada a resposta e recorre ao Módulo Especialista buscando o conteúdo mais adequado à necessidade do aluno, devendo esta estar aderente ao contexto e a intenção selecionada para o tratamento.

Por exemplo, para o caso do ***Aluno A*** o STI deverá considerar que a intenção mais provável será a (i2) ao invés de (i1), visto que ele identificou que o ***Aluno A*** apresenta deficiências no entendimento conceitual sobre operadores condicionais. Logo, usará como resposta uma explicação mais teórica sobre o assunto ao invés de apresentar um exemplo de uso de operador condicional ***IF***, embora fosse uma possibilidade. No momento que o aluno superar essa dificuldade, espera-se que o STI, quando colocado diante da mesma pergunta para esse mesmo aluno, possa indicar um exemplo de sintaxe e semântica do uso de ***IF*** ou que proponha ao aluno a ***Caixa de Comandos Rápidos*** para que ele exercite o uso dos operadores condicionais em pequenos testes.

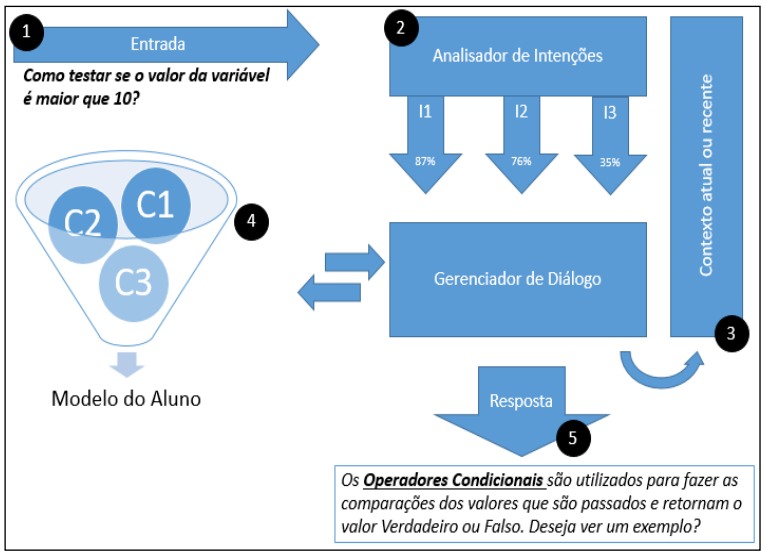

Figura 2 – Processo de decisão de diálogo

5.3. Módulo Especialista

O módulo especialista do STI será composto por duas subdivisões, o módulo de ***Fundamentos de Algoritmos***, que atuará como um organizador do conteúdo didático sobre algoritmos, compreendendo textos, imagens, pseudocódigos e exemplos na linguagem de programação, e o ***Catálogo de Erros,*** que manterá as informações de erros

informados durante a execução do programa e explicações de como evitá-los. Todo material desse módulo possuirá marcações que indicam seu nível, tipo de conhecimento e os pré-requisitos necessários no aluno para que a mensagem seja bem entendida.

5.4. Interface com Usuário

Esse módulo é a principal interface com o aluno e é constituído de outros sub módulos, ***Visualizador de Problemas,*** que será responsável por exibir um determinado problema para o aluno, apontando seu enunciado e exemplos de entrada e saída de dados para cada problema; a ***Caixa de Diálogo***, responsável por fornecer a interface de diálogo com o aluno e a ***Caixa de Comandos,*** onde o aluno poderá executar comando e verificar o resultado imediatamente.

O módulo ***Repositório de Problemas***, que permitirá que o professor catalogue seus exercícios e cada enunciado seja enriquecido com anexos de imagens, pseudocódigos, vídeos e outros recursos multimídia que facilite o entendimento do problema. O professor também poderá mapear os elementos da linguagem que ele espera que sejam utilizados, tais elementos poderão ser visualizados pelo aluno ou poderão ser marcações ocultas onde o STI deverá rastrear o seu uso e registrar como um conhecimento estratégico aplicado na atividade.

O ***Ambiente de Programação***, trata-se de um editor de Python embarcado no STI, onde programas poderão ser criados e executados diretamente no navegador de Internet através do módulo ***Interpretador Python***. Estes recursos permitirão que o aluno inicie o desenvolvimento em qualquer local, inclusive em dispositivos móveis sem a necessidade de instalação de qualquer programa.

6. Técnicas para a Implementação da Arquitetura Prosposta

O trabalho realizado no *iAIML* oferece uma possibilidade de aplicação dessa técnica no módulo ***Analisador de Intenções***. Outra alternativa é uso aplicações online de processamento de linguagem natural, por exemplo: o *Google Dialog Flow*[1], o *Microsoft LUIS* (*Language Understanding Intelligent Service*)[2] e de ferramentas de código-fonte aberto como o Python *NLTK* (*Natural Language Toolkit*)[3] e o *RASA NLU*[4] são outras alternativas para implementação do Analisador de intenções. A vantagem da utilização de ferramentas online é que estas são constituídas de redes neurais já treinadas com milhares de textos sobre variados assuntos, permitindo que as análises das intenções possam capturar diversas nuances idiomáticas. Uma desvantagem é que os textos associados a programação são muitos específicos e poderão escapar à interpretação, exigindo que novos treinamentos sejam realizados para uma coerência nas intenções identificadas.

[1] https://dialogflow.com

[2] https://www.luis.ai/

[3] https://www.nltk.org

[4] https://rasa.com/

Por tratar diversas fontes de dados oriundas dos módulos ***Modelo do Aluno***, ***Analisador de Intenções*** e as informações contextuais fornecidas pela ***Interface Gráfica com o Usuário***, o emprego de redes bayesianas apresenta-se como uma forte alternativa para a implementação ***Gerenciador de Diálogo*** no tratamento da tutoria, tal técnica permite que hipóteses de possíveis cenários sejam criadas a partir das experiências de professores na disciplina, os trabalhos de [Ferreira et al. 2016] e [Vier, Gluz e Jaques 2015] aproximam-se desta proposta. Outro viés é a possiblidade do professor ir refinando o ambiente, enriquecendo-o com novas hipóteses e registrando suas decisões ao responder as dúvidas não atendidas pelo STI. Outras alternativas como o emprego de algoritmos genéticos ou redes neurais poderão ser utilizadas a fim de minimizar a necessidade de um especialista [Miranda, De Melo e Vaz 2017].

Algumas técnicas são vislumbradas para a implementação do **Modelo do Aluno**. Referindo-se ao conhecimento conceitual (C1), devido às incertezas ligadas a tarefa de identificar o nível de conhecimento em linguagem de programação, [Vier, Gluz e Jaques 2015] empregaram o uso das redes bayesianas com objetivo de identificar as probabilidades de um aluno dominar ou não um dado conhecimento. O registro do conhecimento procedural (C2) e o estratégico/analítico (C3) poderão ser implementados através da busca de ocorrências de comandos e padrões no código submetido.

Em [Porfirio, Pereira e Maschio 2017] é identificado o uso de *AST* (*Abstract Syntax Tree*) para representar o código do aluno e então realizar buscas de ocorrências nessa árvore. Uma adequação dessa técnica neste trabalho permitiria o professor registrar pequenos trechos de códigos que ele deseja ver implementado na solução. Então, o código do aluno e os os registros do professor seriam convertidos em *AST* para realização de comparações. Esses pequenos trechos de código do professor não indicariam uma solução padrão para um dado exercício e sim uma forma rápida do professor registrar comandos esperados em dada atividade e o STI avaliar a ocorrência.

O trabalho de [Vier, Gluz e Jaques 2015] permite que inferências sobre o conhecimento do aluno possam ser obtidos através da implementação de redes bayesianas. Uma rede sobre o conhecimento de algoritmos e de lógica de programação foi criada a partir de diversas hipóteses de aprendizagens definidas por professores experientes. Uma possível implementação é alimentar essa rede com evidências de aprendizados obtidos de C1, C2 e C3 e validar esses conhecimentos a partir da reação da rede para cada conhecimento esperado e retroalimentar o modelo do aluno.

As ferramentas *Highlight.js*[5] e *Ace*[6] são boas alternativas para construção do editor online, pois permite replicar comportamentos simples, mas importantes, como o autocomplemento dos comandos, destaque em cores dos sintaxe correta dos comandos e a auto-organização do código.

Para construção do módulo Interpretador Python as ferramentas *Skulp*[7] e *Bryton*[8] são alternativas para a implementação do ambiente de programação diretamente no navegador de Internet sem a necessidade de instalações.

[5] https://highlightjs.org

[6] https://ace.c9.io

[7] http://www.skulpt.org

7. Considerações Finais

Em muitos tutores empregam-se a clássica arquitetura composta principalmente pelo modelo do aluno e o módulo tutor, esses módulos, na prática, pouco têm influenciado na construção do diálogo, por consequência o diálogo não tem sido aderente as necessidades registradas no modelo do aluno e a riqueza dos dados de contexto ao decidir por uma resposta são desconsideradas.

O presente trabalho resultou numa proposta de arquitetura de STI que prover mecanismos direcionados a construção do diálogo com o estudante iniciante em algoritmos e lógica de programação. Propôs-se, uma expansão dos módulos padrões de um STI objetivando o tratamento do diálogo e apontou-se um fluxo de dados que permitirá o tratamento mais eficiente das intenções do aluno ao realizar uma pergunta ou solicitar uma dica antes e durante a realização de atividades no ambiente.

Considerando que a implementação da arquitetura não foi demonstrada nesse estágio da pesquisa, outros trabalhos, ferramentas e técnicas foram referenciadas como possíveis vieses de implementação da arquitetura proposta.

Trabalhos futuros poderão atuar na construção de diálogos mais longos, onde o STI possa propor novas discussões não motivadas pelos alunos. A arquitetura aqui proposta poderá ser incrementada com dados emocionais e de perfis de aprendizados. Outra possiblidade é a criação de um modelo que considere as variadas estratégicas pedagógicas durante a tutoria.

8. Referências

Aureliano, Viviane Cristina O; Tedesco, PC de AR; Giraffa, Lúcia Maria M. Desafios e oportunidades aos processos de ensino e de aprendizagem de programação para iniciantes. In: *Congresso da Sociedade Brasileira de Computação*. 2016.

Bombasar, James et al. Ferramentas para o ensino-aprendizagem do pensamento computacional: onde está Alan Turing?. In: *Simpósio Brasileiro de Informática na Educação-SBIE*. 2015. p. 81.

Coronado, Miguel et al. A cognitive assistant for learning java featuring social dialogue. In: *International Journal of Human-Computer Studies*, 2018.

Das Neves, André Menezes Marques. iAIML: um mecanismo para o tratamento de intenção em Chatterbots. 2005.

De Oliveira, Hilário TA et al. Dr. Pierre: Um Chatterbot com Intenção e Personalidade Baseado em Ontologias para Apoiar o Ensino de Psiquiatria. In: *Simpósio Brasileiro de Informática na Educação-SBIE*. 2010.

El Haddad, Ibrahim A.; NASER, Samy S. Abu. ADO-Tutor: Intelligent Tutoring System for leaning ADO. NET. 2017.

Falckembach, Gilse A. Morgental; de Araujo, Fabrício Viero. Aprendizagem de algoritmos: dificuldades na resolução de problemas. *Anais Sulcomp*, v. 2, 2013.

[8] http://brython.info

Ferreira, Hiran et al. Uma Abordagem Híbrida para Acompanhamento da Aprendizagem do Estudante Baseada em Ontologias e Redes Bayesianas em Sistemas Adaptativos para Educação. In*: Anais dos Workshops do Congresso Brasileiro de Informática na Educação*. 2016. p. 447.

Gerdes, Alex; Heeren, Bastiaan; Jeuring, Johan. Teachers and students in charge. In: *European Conference on Technology Enhanced Learning*. Springer, Berlin, Heidelberg, 2012. p. 383-388.

Júnior, Gps Santos; Fechine, Joseana Macêdo; Costa, E. B. Analogus: Um Ambiente para Auxílio ao Ensino de Programação Orientado pelo Raciocínio por Analogia. *XVII WEI*, v. 28, 2009.

Koliver, Cristian; Dorneles, Ricardo Vargas; Casa, Marcos Eduardo. Das (muitas) dúvidas e (poucas) certezas do ensino de algoritmos. In: *XII Workshop de Educação em Computação*. 2004.

Kölling, Michael et al. The BlueJ system and STI pedagogy. Computer Science Education, v. 13, n. 4, p. 249-268, 2003.

Medeiros, Luciano Frontino de; Junior, Armando Kolbe; Moser, Alvino. THOTH: Um Assistente Cognitivo com Small Talks para Conversação Tutorial. In: *Simpósio Brasileiro de Informática na Educação-SBIE*. 2017. p. 1799.

Moreira, Gabriel Luídy et al. Desafios na aprendizagem de programação introdutória em cursos de TI da UFERSA, campus Pau dos Ferros: um estudo exploratório. *Anais do Encontro de Computação do Oeste Potiguar ECOP/UFERSA*, v. 2, n. 1, 2018.

Paz, Fábio J. et al. Perspectivas tecnológicas para o aprimoramento de chatbots educacionais em AIML. TE & ET, 2017.

Pinheiro, Diego et al. Exploração do uso de bases de conhecimento e processamento de linguagem natural em um simulador de casos clínicos. *Anais do Computer on the Beach*, p. 641-650, 2018.

Porfirio, Andres; Pereira, Roberto; Maschio, Eleandro. Atualização do Modelo do Aprendiz de Programação de Computadores com o Uso de Parser AST. In: *Anais dos Workshops do Congresso Brasileiro de Informática na Educação*. 2017.

Price, Thomas W.; Dong, Yihuan; Lipovac, Dragan. iSnap: towards intelligent tutoring in novice programming environments. In: *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education*. ACM, 2017.

Tobar, Carlos Miguel et al. Uma arquitetura de ambiente colaborativo para o aprendizado de programação. In: *Simpósio Brasileiro de Informática na Educação-SBIE*. 2001. p. 367-376.

Ueno, Haruki. INTELLITUTOR: A knowledge based intelligent programming environment for novice programmers. In: *COMPCON Spring'89. Thirty-Fourth IEEE Computer Society International Conference.* IEEE, 1989. p. 390-395.

Vier, Juliano; Gluz, João; Jaques, P. Empregando redes bayesianas para modelar automaticamente o conhecimento dos aprendizes em lógica de programação. *Revista Brasileira de Informática na Educação*, v. 23, n. 2, 2015.