UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO

PROGRAMA DE PÓS-GRADUÇÃO EM INFORMÁTICA

RENATO DE MOURA SANTOS

UM AMBIENTE ONLINE PARA APOIO DIALOGADO À APRENDIZAGEM DE PROGRAMAÇÃO

VITÓRIA – ES, BRASIL
2019

RENATO DE MOURA SANTOS

UM AMBIENTE ONLINE PARA APOIO DIALOGADO À APRENDIZAGEM DE PROGRAMAÇÃO

Dissertação apresentada ao Programa de Pós-Graduação em Informática do Departamento de Informática da Universidade Federal do Espírito Santo, como requisito parcial para obtenção do título de Mestre em Informática, na área de concentração de Informática na Educação.

Orientador: Prof. Dr. Crediné Silva de Menezes

VITÓRIA – ES, BRASIL
2019

Ficha catalográfica disponibilizada pelo Sistema Integrado de Bibliotecas - SIBI/UFES e elaborada pelo autor

---

DE MOURA SANTOS, RENATO, 1981-
D278a UM AMBIENTE ONLINE PARA APOIO DIALOGADO À APRENDIZAGEM DE PROGRAMAÇÃO / RENATO DE MOURA SANTOS. - 2019.
155 f. : il.

Orientador: Crediné Silva de Menezes
.
Dissertação (Mestrado em Informática) - Universidade Federal do Espírito Santo, Centro Tecnológico.

1. Linguagem de programação (Computadores). 2. Agentes inteligentes (Software). 3. Inteligência artificial - Aplicações educacionais. 4. Informática na educação. I. Silva de Menezes, Crediné. II. Universidade Federal do Espírito Santo. Centro Tecnológico. III. Título.

CDU: 004

RENATO DE MOURA SANTOS

UM AMBIENTE ONLINE PARA APOIO DIALOGADO À APRENDIZAGEM DE PROGRAMAÇÃO

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Informática da Universidade Federal do Espírito Santo, como requisito parcial para obtenção do Grau de Mestre em Informática.

Aprovado em 22 de julho de 2019.

_____
Prof. Dr. Crediné Silva de Menezes
Programa de Pós-Graduação em Informática – UFES
Faculdade de Educação – UFRGS
Orientador

_____
Prof. Dr. Davidson Cury
Programa de Pós-Graduação Informática - UFES

_____
Prof. Dr. Alberto Nogueira de Castro Junior
Programa de Pós-Graduação em Informática - UFAM

Brasil
2019, Julho

Dedico esse trabalho aos meus avós, Maria de Moura (in memoriam) e Germano de Moura (in memoriam) que me apresentaram a fé e os primeiros livros, à minha esposa, Fabiola, e as minhas filhas, Maria Loisa e Sophia.

AGRADECIMENTOS

Rendo meus primeiros agradecimentos ao Senhor, a quem devo a minha vida e a força necessária para vencer.

Ninguém faz alguma coisa ou se torna alguém sozinho. A cooperação é a coisa mais elegante e fundamental da raça humana. Portanto, agradeço a todos os gigantes que me cederam seus ombros ou que me mostraram o caminho das pedras, o valor da persistência e da fé.

Á minha esposa, Fabiola, agradeço pelo amor, compreensão e companheirismo.

Às minhas filhas Maria Loisa e Sophia, rendo meus agradecimentos pela paciência durante todo o tempo de dedicação a esse trabalho, espero também ensiná-las com essa conquista.

Aos meus pais que me ensinaram os valores de integridade e de persistência. E por desde cedo me mostrarem que a educação era o caminho para transformação pessoal e social.

Presto meus agradecimentos ao meu orientador Dr. Crediné Silva de Menezes pelos riquíssimos momentos que levarei para toda vida. Sua orientação além de me instigar na busca da excelência da pesquisa também me mostrou que a ciência e a educação são juntas uma importante ferramenta para melhorar a condição humana.

Aos colegas Ramon Ahnert Azeredo, Marcos Lovati, Izon Thomaz Mielke, Francisca Souza e Jamilli Ricarto agradeço pelo apoio, oportunidades de compartilhamento e pelas experiências trocadas.

Agradeço também ao Gabriel Santos (meu grande irmão) e Bianca por seus empenhos oferecidos nesse trabalho.

Por último e não menos importante, meu muito obrigado ao Alysson Frizzera (meu coordenador de curso no CEET-Vasco Coutinho) por sua compreensão e apoio durante essa caminhada.

“Em algum lugar, alguma coisa incrível está esperando para ser conhecida.’

Carl Sagan

“É como está escrito: Coisas que os olhos não viram, nem os ouvidos ouviram, nem o coração humano imaginou (Is 64,4), tais são as coisas que Deus tem preparado para aqueles que o amam.’

1 Coríntios 2:9

RESUMO

O ensino da disciplina de algoritmos e linguagem de programação têm se tornado cada vez mais importante num mundo conectado pelas tecnologias da informação e a sua aprendizagem é essencial para as demais disciplinas num curso de computação, uma vez que a resolução de problemas através de algoritmos ocorre em praticamente todo o curso. Embora a importância do ensino de programação, as dificuldades de aprendizagem desta disciplina são notórias e o processo de ensino apresenta diversos desafios ao professor e ao aluno. As principais dificuldades dos alunos são evidenciadas durante a realização das atividades propostas e residem principalmente no desenvolvimento da lógica de programação, na falta de domínio da sintaxe e semântica da linguagem de programação e dificuldades na interpretação dos problemas propostos. Por outro lado, ensinar a programação para alunos iniciantes exige do professor uma forte demanda de interação a fim de atender, acompanhar, mediar e avaliar estes alunos. Esse trabalho apresenta um ambiente de programação com suporte à conversação com o aluno e tem como objetivo oferecer as condições para que este aluno obtenha *feedbacks* durante a resolução dos problemas propostos, ao tempo que libera o professor das demandas recorrentes de atendimentos. Portanto, foi realizado um estudo de caso e pesquisa de referencial teórico a fim de se conhecer as dificuldades recorrentes que poderiam ser transferidas para o ambiente e para identificar as abordagens computacionais que poderiam ser utilizadas na implementação de uma proposta de ambiente. Foram criados os assistentes pedagógicos baseados em agentes inteligentes responsáveis por efetuarem atendimentos às dificuldades informadas pelos alunos ou dedicados à análise do código-fonte das atividades desenvolvidas em busca de indícios de dificuldades não expressadas diretamente. A validação da proposta ocorre com a implementação de uma prova de conceito de parte da solução e a aplicação deste protótipo a curso de intensivo de introdução à linguagem de programação. Os dados coletados foram analisados e os resultados obtidos apresentaram evidências que a solução proposta fornece as condições para melhorar o suporte ao aluno durante a realização das atividades, além de liberar o professor dos casos recorrentes dos pedidos de atendimentos.

Palavras-chave: Introdução à Programação, Importância do *Feedback*, Dificuldades na Aprendizagem de Programação, Sistemas Multiagentes,

Recuperação da Informação.

ABSTRACT

Teaching the discipline of algorithms and programming language has become increasingly important in a world connected by information technology, and learning is essential for the other disciplines in a computer course, as algorithmic problem solving occurs. practically the entire course. Although the importance of programming teaching, the learning difficulties of this subject are notorious and the teaching process presents several challenges to the teacher and the student. The main difficulties of the students are evidenced during the accomplishment of the proposed activities and lie mainly in the development of the programming logic, lack of mastery of the programming language syntax and semantics and difficulties in the interpretation of the proposed problems. On the other hand, teaching programming for beginning students requires a strong demand for interaction from the teacher to attend, monitor, mediate and evaluate these students. This paper presents a programming environment that supports conversation with the student and aims to provide the conditions for this student to obtain feedback during the resolution of the proposed problems while freeing the teacher from the recurring demands of attendance. Therefore, a case study and theoretical reference research were conducted to know the recurrent difficulties that could be transferred to the environment and to identify the computational approaches that could be used in the implementation of an environment proposal. Intelligent agent-based pedagogical assistants were created responsible for attending to the difficulties reported by the students or dedicated to the analysis of the source code of the developed activities in search of signs of difficulties not directly expressed. The validation of the proposal occurs with the implementation of a proof of concept of part of the solution and the application of this prototype to an intensive course of introduction to the programming language. The collected data were analyzed and the obtained results showed pieces of evidence that the proposed solution provides the conditions to improve the support to the student during the accomplishment of the activities, besides releasing the teacher of the recurrent cases of the requests of attendance.

Keywords: Introduction to programming, Providing feedback, Novice

Programmer Difficulties, Multiagents Systems, Information Retrieve.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE SIGLAS

ACs	Assistentes Conversacionais
ACOs	Assistentes Avaliadores de Códigos
AST	Árvore Sintática Abstrata
AIIP	Ambiente Inteligente para Iniciantes em Programação
BOW	Bag of Words
EAD	Educação a Distância
GWT	Google Web Toolkit
HTTP	Hypertext Transfer Protocol
IDF	Inverse Document Frequency item[JSON] JavaScript Object Notation
KNN	K-Nearest Neighbors
MNB	Multinomial Naive Bayes
MVC	Model-View-Controller
NLU	Natural Language Processing
ORM	Object Relational Mapper
PHP	Hypertext Preprocessor
POO	Programação Orientado a Objetos
MVC	Model-View-Controller
RI	Recuperação da Informação
RBC	Raciocínio Baseado em Casos
SRI	Sistema de Recuperação da Informação
STI	Sistemas Tutores Inteligentes
SOA	Service Oriented Architectures

SUMÁRIO

1. Introdução

Este capítulo apresenta uma visão geral da pesquisa desenvolvida, o contexto em que ela se insere, suas motivações, hipóteses, questões que nortearam o seu desenvolvimento, assim como os objetivos esperados, os processos adotados, as contribuições e as produções científicas realizadas. Esta introdução guiará todos os demais capítulos.

O capítulo está organizado nas seguintes seções: a Seção 1.1 apresenta o contexto em que esta pesquisa está inserida. A Seção 1.2 apresenta o problema de pesquisa abordado. Em seguida, na Seção 1.3 são apresentados os objetivos a serem alcançados no decorrer desta pesquisa e na Seção 1.4 são elencadas as questões que nortearão o seu desenvolvimento. Na Seção 1.5 são levantadas as hipóteses que serão validadas nesta pesquisa e na Seção 1.6 são apresentadas as produções cientificas publicadas durante o processo de pesquisa. Por último, a Seção 1.8 apresenta como está organizada esta dissertação.

1.1. Contextualização

O estudo de programação é de suma importância dado que a criação de algoritmos constitui a base das mais variadas tecnologias da informação e a sua influência permeia nas diversas áreas. Além do mais, a aprendizagem de programação é benéfica para o desenvolvimento de habilidades cognitivas necessárias para a resolução de problemas, inclusive sendo atualmente uma disciplina ensinada para crianças e adolescentes em muitas instituições no Brasil e no mundo.

Embora a relevância do estudo da disciplina de programação, apreender a programar é uma tarefa complexa e que apresenta inúmeros desafios para os alunos iniciantes, uma vez que estes estudantes devem aprender a solucionar problemas algoritmicamente, aprender uma nova linguagem (sintaxe e semântica), utilizar um ambiente de programação, além de testar e depurar programas desenvolvidos (JÚNIOR; FECHINE; COSTA, 2009; BOULAY, 1986). Tais dificuldades podem ser diagnosticadas pelo alto grau de repetência

na disciplina e pelas dificuldades dos alunos noutras disciplinas que exigem o domínio dessas habilidades como pré-requisito (TOBAR et al., 2001).

1.3. Objetivos

O objetivo dessa pesquisa é propor um ambiente de ensino-aprendizagem concebido para dá suporte a autoria de programas, monitoramento, análise e a atendimento às dificuldades dos alunos iniciantes em linguagem de programação e também propor abordagens que permitam minimizar o esforço e o tempo dedicado pelo professor no atendimento das demandas desses alunos ocorridas durante a realização das atividades propostas na disciplina.

Para alcançar o objetivo principal serão considerados os seguintes objetivos específicos:

- Propor um modelo conceitual que permita a extensão das funcionalidades propostas e a criação de novos recursos no ambiente;
- Prover ao aluno um ambiente para executar as atividades de programação online sem a necessidade de instalações, limitações geográficas ou temporais;
- Desenvolver um ambiente computacional que apoie o aluno em suas dificuldades durante a realização das atividades propostas;
- Desenvolver um ambiente computacional que estenda a capacidade de atendimento do professor e permitir que ele acompanhe, apoie e avalie as atividades realizadas pelo aluno no ambiente;
- Identificar as abordagens computacionais e as ferramentas disponíveis que viabilizem a implementação do ambiente numa prova de conceito.

1.2. Problema de Pesquisa

O ensino de programação possui uma forte demanda de interação a fim de atender, acompanhar, mediar e avaliar individualmente os alunos e suas atividades (RAABE; SILVA, 2005b). Por outro lado, Aureliano, Tedesco e

Giraffa (2016) afirmam que é extremamente difícil para um professor atender todas as demandas dos alunos ocorridas durante a realização das atividades e esse problema torna-se mais agravado quando o estudante se encontra fora dos limites de sala de aula.

O problema a ser investigado é identificar como prover um ambiente que propicie suporte ao aluno iniciante em linguagem de programação de forma que este ambiente otimize o tempo do professor no atendimento das dificuldades ocorridas durante a realização exercícios propostos.

1.4. Questões Norteadoras

- Como diminuir a necessidade de intervenção do professor das interações básicas e recorrentes durante as atividades de programação?
- Quais são os contributos computacionais empregados no auxílio as dificuldades do aluno iniciante em programação?
- Como criar um modelo arquitetural que flexibilize a implementação de novas funcionalidades no ambiente proposto?
- Quais recursos computacionais podem viabilizar a construção de uma prova de conceito?

1.5. Hipóteses

A formulação das hipóteses da pesquisa pode ter seu embasamento nas análises do conhecimento científico disponível, podendo ser oriundas de outras pesquisas (SILVA; MENEZES, 2001). As hipóteses elaboradas com base nos resultados de outras investigações geralmente conduzem a conhecimentos mais amplos que aquelas decorrentes da simples observação (GIL, 2009). Com base neste entendimento metodológico, esta pesquisa fundamenta-se nas seguintes hipóteses:

- Os alunos iniciantes em programação constantemente cometem erros, descobrem dúvidas e expressam a necessidade de ajuda imediata para avançarem, entretanto, nem sempre o docente está disponível para ajudá-los (CARTER; DEWAN; PICHILIANI, 2015; MOTA et al.,2009;

KOLIVER; DORNELES; CASA,2004). O uso de suporte computacional que forneça *feedback* automático mostra-se uma solução para apoiar professor no atendimento de demandas recorrentes e favorece o aprendizado do aluno ao propiciar suporte no momento de suas dificuldades;

- A conversação é uma importante funcionalidade empregada na interação humano- computador (IHC). No contexto do ensino de programação uma interface em língua natural motiva a participação dos aprendizes e desempenha um papel importante no aprendizado (TOBAR et al., 2001) e esse tipo de interação permite que os alunos se relacionem de forma mais humana com o computador (CORONADO et al., 2018). Portanto, um ambiente educacional que ofereça a interação do aluno via conversação permite que o aprendiz expresse as suas dificuldades e receba o suporte no tempo adequado;
- Um ambiente de codificação pode ser acompanhado por sistemas de agentes inteligentes responsáveis por monitorarem o desenvolvimento das atividades propostas e então interagirem com o aluno fornecendo-lhe suporte nas dificuldades ocorridas.

1.6. Produção Científica

Os seguintes artigos foram produzidos e publicados durante esta pesquisa:

- SANTOS, Renato; DE MENEZES, Crediné; CURY, Davidson. Uma Arquitetura de Tutor Inteligente que Provê Suporte ao Diálogo com o Aluno Iniciante em Linguagem de Programação. In: Anais dos Workshops do Congresso Brasileiro de Informática na Educação. 2018. p. 768.

- SANTOS, Renato de M.; DE MENEZES, Crediné; CURY, Davidson. TutorDevChat, an intelligent tutor who dialogues with the novice programmer during learning and solving exercises. In: Workshop Latino-Americano de Trabalhos em Andamento em Computação (WLATAC), 2018.

1.7. Metodologia

Neste capítulo será apresentada a metodologia de pesquisa e os procedimentos metodológicos utilizados no desenvolvimento desta dissertação.

1.7.1. Metodologia de Pesquisa

A classificação da pesquisa desenvolvida baseia-se nas proposições de Silva e Menezes (2001) e Gil (2009), que estabelecem que uma pesquisa científica pode ser classificada segundo a sua natureza, do ponto de vista de seus objetivos, da forma de abordagem do problema e sob a ótica dos procedimentos técnicos que serão realizados.

Esta pesquisa é de natureza aplicada, uma vez que reunindo um corpo de conhecimento e ferramentas tecnológicas busca-se a aplicação prática em ambiente real na solução de um problema e a avaliação das consequências desta intervenção (SILVA; MENEZES, 2001).

Quanto aos objetivos, a presente pesquisa pode ser classificada como exploratória visto que se buscará uma visão mais ampla e aprofundada da área a ser estudada através de revisão bibliográfica e de sua análise, identificando as dificuldades de alunos e professores no processo de ensino-aprendizagem de programação, bem como os métodos que poderão ser utilizados na busca de uma solução.

Do ponto de vista da abordagem do problema a presente pesquisa é abordada na forma qualitativa, uma vez que não busca quantificar dados, mas sim, impactar qualitativamente no apoio a abordagens pedagógicas dos professores e no apoio aos alunos iniciantes em programação.

Por último, a pesquisa é classificada como pesquisa bibliográfica por ser realizada a partir da identificação do tema do trabalho através da consulta a livros, artigos de periódicos, teses e dissertações defendidas. Também é estudo de caso uma vez que busca explorar as situações reais vivenciadas em sala de aula, validar as hipóteses estabelecidas e os impactos da intervenção realizada.

1.7.2. Procedimentos Metodológicos

Conforme ilustra a Figura 1, os procedimentos metodológicos da pesquisa serão realizados em 6 fases.

Na fase de planejamento será realizada a delimitação do escopo dessa pesquisa assim como definir uma primeira versão de seus objetivos. Ainda nessa fase será planejada a atuação do pesquisador no estudo de caso, essa fase se repetirá em refinamentos sucessivos à medida que for o problema de pesquisa for aprofundado.

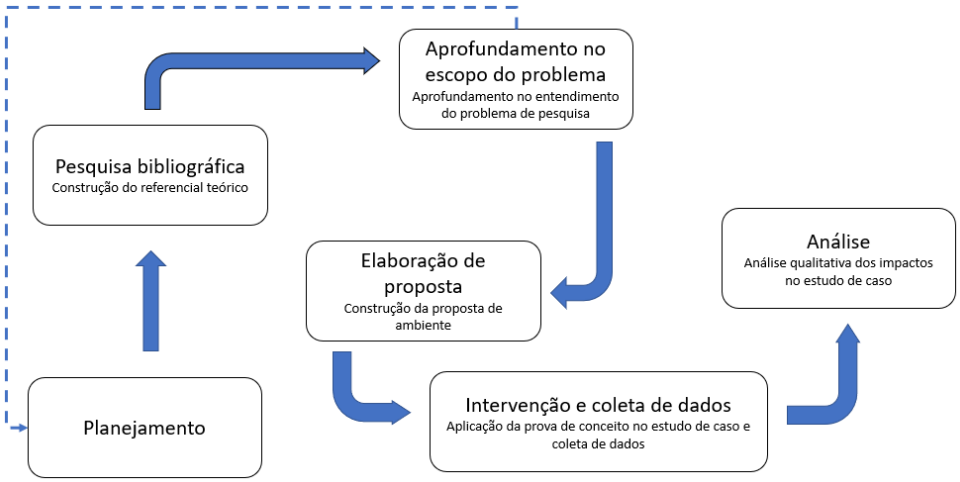


Figura 2 - Síntese do processo de pesquisa.

Fonte: Autoria Própria

Na fase de pesquisa bibliográfica será realizado um levantamento da literatura a respeito do problema de pesquisa e das abordagens computacionais aplicáveis na construção de um ambiente.

Em seguida será realizada a fase de aprofundamento no entendimento do problema onde serão analisados outros trabalhos que abordaram de alguma o problema de pesquisa e as intervenções já realizadas. Esta fase poderá orientar a revisitação da fase de planejamento e de pesquisa bibliográfica.

Na fase de elaboração será apresentada a proposta de ambiente dentro do escopo definido para o tratamento do problema de pesquisa.

A próxima fase refere-se à intervenção num caso real através da inserção da prova de conceito do ambiente proposto. Durante essa fase ocorrerá a coleta de dados através de questionários que serão disponibilizados para que os alunos envolvidos possam avaliar qualitativamente o ambiente e a metodologia aplicada. Também nesta fase serão coletados os dados de logs do ambiente para análise.

Por último, serão analisados os dados coletados e os resultados e impactos identificados serão apresentados.

1.8. Organização do Trabalho

Este trabalho está dividido em 9 capítulos, sendo o primeiro dedicado à introdução onde será apresentada a definição do problema tratado, o objetivo, as questões norteadoras, as hipóteses, a produção cientifica realizada durante período dessa pesquisa, a metodologia aplicada e esta seção.

No Capítulo 2 será apresentado o contexto do problema de pesquisa identificado, onde serão o processo de ensino-aprendizagem de programação e as dificuldades dos professores e dos alunos iniciantes. Ainda nesta seção, serão abordadas a importância do diálogo e do acompanhamento do aluno iniciante em programação durante a resolução de problemas propostos.

No Capítulo 3 será abordada a revisão da literatura sobre os sistemas agentes inteligentes, a recuperação da informação e sobre os padrões de software.

No Capítulo 4 serão apresentados os trabalhos correlatos a este e que objetivaram oferecer o suporte aos alunos iniciantes em programação durante a realização dos exercícios propostos ou que buscavam apoiar o professor durante as dificuldades decorridas neste processo.

No Capítulo 5 será apresentada, primeiramente, o escopo do problema tratado neste trabalho e uma visão geral do ambiente proposto e em seguida as funcionalidades que serão oferecidas aos professores e alunos em atendimento as suas dificuldades durante as práticas de exercícios de programação. Por fim, será apresentado o modelo conceitual proposto.

Já no Capítulo 6 serão apresentadas as tecnologias, frameworks, linguagens de programação utilizadas e outros elementos que darão suporte a implementação do ambiente proposto.

Em seguida, no Capítulo 7 será apresenta uma Prova de Conceito do ambiente proposto onde serão detalhadas as principais implementações e as interfaces ligadas ao ambiente do professor e do aluno no uso das funcionalidades desenvolvidas.

Já no Capítulo 8 será apresentado o experimento realizado com alunos iniciantes na disciplina de algoritmos e lógica de programação, onde o ambiente foi utilizado no suporte ao professor e aos alunos na realização das atividades propostas. Também neste capítulo será realizada a avaliação dos dados qualitativos e quantitativos obtidos durante o experimento.

Finalmente, no Capítulo 9 será realizada uma análise geral dos resultados alcançados em relação aos objetivos e hipóteses estabelecidas no início dessa pesquisa. Ainda neste capítulo serão apresentadas as propostas de trabalhos futuros.

2. Caracterização do Problema

Neste capítulo será contextualizado o problema de pesquisa abordado e os conceitos considerados fundamentais para o desenvolvimento deste trabalho.

Na Seção 2.1 será tratado o processo de ensino-aprendizagem de lógica de programação destacando sua importância e as estratégias pedagógicas ligadas a esse processo.

Em seguida, na Seção 2.2 serão apresentadas as principais dificuldades dos alunos iniciantes em programação.

Já na Seção 2.3 serão apresentadas as dificuldades no processo de ensino de programação sob a perspectiva do professor.

Por último, na Seção 2.4 será abordada a importância do diálogo com o aluno durante a resolução a resolução das atividades de programação.

2.1. O Processo de Ensino-Aprendizagem de Introdução à Linguagem de Programação

O ensino de programação de computadores tem por objetivo a capacitação dos alunos no desenvolvimento de programas e sistemas computadorizados que objetivam a resolução de problemas (PIMENTEL; OMAR, 2008) e sua aprendizagem é essencial para as demais disciplinas num curso de computação, uma vez que a resolução de problemas através de algoritmos ocorre em praticamente todo o curso.

Mesmo considerando a importância do ensino de programação, a dificuldade de os estudantes aprenderem esta disciplina é notória (GOMES et al., 2015; BOULAY, 1986) e apresenta diversos desafios (JÚNIOR; FECHINE; COSTA, 2009). A pesquisa recente de Moreira et al. (2018) aponta que as dificuldades ligadas à programação, já identificadas há décadas, ainda persistem e continuam 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 a dedicação na disciplina (26.36%) e a dificuldade na interpretação das questões (15.45%).

2.1.1. Estratégias Pedagógicas no Ensino de Programação

O aprendizado de programação é constituído de uma tarefa complexa que inclui: adquirir a habilidade para resolução de problemas, aprender a sintaxe e a semântica de uma linguagem de programação, utilizar um ambiente de programação e realizar testes e depuração de programas (BOULAY, 1986; DELGADO et al., 2005). Pesquisadores apontam que solucionar problemas por meio de algoritmos é um dos desafios que apresenta o maior grau de dificuldade para os alunos iniciantes (DELGADO et al., 2005; MOREIRA et al., 2018) já que eles sentem dificuldades em analisar problemas e em formular soluções.

Parte das dificuldades no processo de ensino-aprendizagem de programação são oriundas da falta de um método de ensino adequado e devido a aplicação de pressupostos pedagógicos que, no ensino desse conteúdo, partem do abstrato para o concreto (RAPKIEWICZ et al., 2007). No ensino de programação ainda é muito aplicado o método tradicional no qual o professor expõe os conceitos e exemplos com os clássicos quadro e pincel. Embora existam outras modalidades de ensino como a EAD e o ensino semipresencial, prevalece na abordagem desta disciplina um estilo conteudista, centrada no professor como transmissor do conhecimento e o aluno como o receptor (HOLANDA; COUTINHO; FONTES, 2018). Na avaliação das atividades é comum que professor apresente uma solução modelo e, indiretamente, busca-se que o aluno se adeque à forma de resolução prevista pelo docente (GIRAFFA; MARCZAK; ALMEIDA, 2003; JUNIOR; FRANÇA, 2017). Também persiste o tratamento dos alunos no coletivo (a turma), não sendo respeitado o contexto da ocorrência da dificuldade, o ritmo e as motivações de cada indivíduo (PIMENTEL; OMAR; FRANÇA, 2005).

Uma metodologia de ensino-aprendizagem deveria, primeiramente, dar condições ao aluno e professor para identificar as dificuldades ocorridas e seguida instituir um ensino pautado nas necessidades individuais de aprendizagem (PIMENTEL; OMAR, 2008). Pimentel et al. (2004) orienta que a identificação do que se sabe e do que não se sabe pode ser obtido através de

avaliações contínuas, integradas ao processo de aprendizagem e não apenas em momentos finais.

É observado que a maior parte do tempo desprendido ao ensino de programação é dedicado a resoluções de problemas propostos, ao atendimento as dúvidas e às dificuldades ocorridas durante essas atividades. Entretanto, é identificado que essas atividades propostas são muitas das vezes atreladas a problemas matemáticos e, por conseguinte, são abstratas em seus enunciados e distante da realidade vivenciada pelo aluno (PIEKARSKI et al., 2015) e dessa forma não é aproveitado o conhecimento prévio do aprendiz.

Outra dificuldade está na identificação de um paradigma de linguagem de programação que seja mais adequado ao aluno iniciante. Na computação existem diversos paradigmas de programação, dentre eles o funcional, o procedural (ou imperativo), declarativo e o de programação orientado a objetos (POO). Essa variedade de paradigmas, embora importante nas diferentes abordagens para a resolução de problemas computacionais, gera o problema de se definir qual o melhor paradigma a ser utilizado no ensino introdutório de programação (KOLIVER; DORNELES; CASA, 2004). A POO é defendida por permitir resolução com alto grau de reusabilidade, todavia, é necessário que aluno aprenda conceitos como Classes, Objetos, Referências dentre outros aspectos desse paradigma (KÖLLING et al., 2003).

Do lado oposto, há o argumento que o processo de ensino de algoritmos deve, em essência, focar na resolução de problemas e aproveitar ao máximo o que o aluno já sabe, evitando abstrações, paradigmas e detalhes da linguagem de programação, como se observa nos ambientes como o App Inventor (WOLBER, 2011) e Scratch (RESNICK et al., 2009) que são amplamente utilizados com este propósito.

Falckembach e Araujo (2013) argumentam que o paradigma procedural é eficiente no ensino, pois, facilita ao aluno compreender e explicitar as ações que compõem um algoritmo devido ao seu sequenciamento nas instruções fornecidas.

A escolha de uma linguagem de programação deve ser baseada na sua adequação aos objetivos finais com o aluno e somente deve ser escolhida após o estabelecimento das metas do curso e dos resultados de aprendizagem terem sidos especificados (PEARS et al., 2007), todavia ainda não há com consenso sobre qual linguagem ou paradigma é mais adequando ao ensino introdutório de programação.

2.2. As Dificuldades do Aluno Iniciante em Programação

Os alunos recém-chegados na disciplina de programação estão habituados às disciplinas nas quais é possível ser bem-sucedido através de abordagens de estudo baseadas em leituras sucessivas, memorização de fórmulas e uma certa mecanização de procedimentos (GOMES; HENRIQUES; MENDES, 2008). Todavia, a aprendizagem de programação impõe um estudo bastante diferente, exigindo uma prática intensiva, a habilidade para resolução de problemas, a aprendizagem da sintaxe e da semântica de uma linguagem de programação, a utilização um ambiente de programação e a realização de testes e depuração de programas (BOULAY, 1986).

As dificuldades dos alunos no aprendizado de programação são evidentes pelos altos índices de reprovação e o baixo desempenho destes alunos noutras matérias que possuem a programação como base (TOBAR et al., 2001). Ao iniciar os estudos de algoritmos os estudantes encontram um obstáculo muito grande em aplicar suas habilidades prévias, criando uma fonte de medo e frustração (IEPSEN, 2013). As consequências diretas são as reprovações sistemáticas, a apatia com a disciplina, a baixa autoestima e desistência da disciplina, culminando na maioria das vezes no abandono do curso (DELGADO et al., 2005).

Raabe e Silva (2005) classificaram os aspectos que contribuem para a dificuldade de aprendizagem nos alunos iniciantes em programação em três tipos. São eles:

- Problemas de natureza didática: São aqueles que envolvem a diversidade de cultura e de experiência dos alunos ingressantes, a excessiva quantidade de alunos por turma, o que inviabiliza um

atendimento e avaliação individual, a dificuldade de expressar e compreender a lógica desenvolvida e a ausência de materiais de referência de qualidade aos alunos. Além disso, têm-se os problemas particulares, que dizem respeito ao desconhecimento do aluno sobre o perfil do curso;

- Problemas de natureza cognitiva: Consiste em aspectos que normalmente estão atrelados a vivência precedente ao ingresso no curso. Por exemplo, são encontrados em alunos cujos estudos foram interrompidos há algum tempo, ou que o ensino médio não os permitiu o desenvolvimento adequadamente as faculdades cognitivas e alunos que já possuíam muitas dificuldades em lógica e matemática;
- Problemas de natureza afetiva: Diz respeito aos fatores emocionais ocasionais, ou problemas constantes que se manifestam durante todo o decorrer da disciplina em maior ou menor grau, como a baixa autoestima, a pouca motivação, a aversão ao conteúdo ou ao docente, a insegurança e etc.

2.3. As Dificuldades do Professor no Ensino de Programação

Na sala de aula, independentemente da disciplina, o professor depara-se com alunos apresentando graus de conhecimento e habilidades heterogêneas, ou seja, alunos com perfis distintos de aprendizagem, de motivação e de ritmo, isto não é diferente em disciplinas introdutórias de algoritmos e lógica de programação (NOBRE; MENEZES, 2002; RAABE; SILVA, 2005a).

Ensinar programação para alunos iniciantes, exige do professor uma forte demanda de interação a fim de atender, acompanhar, mediar e avaliar os alunos. No entanto, em muitas situações esta demanda de interação é inviável de ser atendida devido a quantidade de alunos e a pluralidade de dificuldades de aprendizagem apresentadas por estes alunos (CASTRO; JÚNIOR; MENEZES, 2004). Frequentemente, os professores esquecem-se de diversificar as suas estratégias de forma a contemplar a multiplicidade de pensamentos, compreensões, ritmos e estilos de aprendizagem existentes em cada turma acarretando dificuldades nestes alunos (GOMES; HENRIQUES; MENDES, 2008).

França e Soares (2011) observam que para o professor retirar uma simples dúvida de um aluno é necessário, no mínimo, que ele se desloque até o aluno, observe a execução do programa e verifique o seu resultado. Numa turma numerosa essa atividade de simples verificação pode tornar o tempo de aula insuficiente (FALCKEMBACH; ARAUJO, 2013; FRANÇA; SOARES, 2011) ou acarretar em diversos alunos sem atendimento requerido.

2.4. A Importância do Diálogo e do Acompanhamento do Aluno durante as Atividades de Programação

É fundamental oferecer condições aos educandos em suas socializações com os seus pares e com o professor. Um ambiente de aprendizagem deve propiciar ao aluno testar a sua experiência de assumir-se como um ser histórico e social, que pensa, que critica, que opina, que tem sonhos, que se comunica e que dá sugestões (FREIRE, 1987). Neste sentido a comunicação oportunizada pelo diálogo permite que tanto o professor quanto o aluno, cresçam juntos e estabelecem uma relação mais afetiva, permitindo que o ensino aconteça de forma prazerosa e mais significativa, Freire (1997) ressalta que “o educador já não é o que apenas educa, mas o que, enquanto educa, é educado, em diálogo com o educando que, ao ser educado, também educa” e destaca que é bom lembrar que ele (o professor) precisa sempre estar atento à curiosidades e perguntas dos alunos, devendo sempre ensinar a aprender e não só transferir conhecimento.

No contexto da aprendizagem significativa, Moreira (2006) defende que a aprendizagem ocorre no domínio de interações perturbadoras que geram mudanças de estado. Neste sentido, os novos conhecimentos são perturbações que, na aprendizagem significativa, receberão significados e, ao mesmo tempo, através de uma interação perturbadora, se modificarão em alguma medida a estrutura dos conhecimentos prévios sem alterar sua organização. Dentro dessa perspectiva, cabe observar que as respostas apenas devem surgir quando houver questionamentos e não ao contrário (SURMACZ; LOPES, 2018), pois para que aconteça o conhecimento é necessário que o sujeito tenha se dado conta de seu não saber sobre algo, quando então uma pergunta se faz emergente (SANTOS, 2012). Entende-se,

portanto, que o uso da pergunta como estratégia de ensino, reúne condições para que os conteúdos disciplinares trabalhados em sala de aula tornem-se interessantes e significativos na percepção do aluno (SURMACZ; LOPES, 2018).

Um elemento substancial oriundo do diálogo é o *feedback*, ele é conceituado como as informações fornecidas por um agente, por exemplo, o professor, um colega, o livro ou a própria reflexão aluno sobre um dado assunto. O *feedback* ocorre em relação aos aspectos do desempenho ou da compreensão de uma pessoa (HATTIE; TIMPERLEY, 2007) e pode ter diferentes funções: reconhecer, confirmar, reforçar a resposta correta ou resultados de aprendizagem de alta qualidade, e promover a aquisição do conhecimento e das operações cognitivas necessárias para a realização da aprendizagem. Além disso, no nível motivacional, o *feedback* pode incentivar os alunos a manterem o esforço e a persistência (NARCISS, 2013). Por exemplo, um professor ou pai pode fornecer informações corretivas, um colega pode fornecer uma estratégia alternativa, um livro pode fornecer informações para esclarecer ideias, um pai pode fornecer incentivo e o próprio aluno pode procurar informações para validar a exatidão de uma resposta (HATTIE; TIMPERLEY, 2007).

No contexto da educação uma interface com língua natural motiva a participação dos aprendizes e desempenha um papel importante no aprendizado (SANTOS; MENEZES; CURY, 2018), no entanto, um *feedback* individual pode consumir muito tempo do professor com o risco de que outros estudantes possam não se beneficiar dele no devido tempo (QUEIRÓS; LEAL, 2015).

Ao promover o diálogo e, por conseguinte a pergunta do aluno, o professor introduz no ambiente de ensino as condições de promoção da interação entre os sujeitos que, ao questionarem, exibem suas inquietações compreendidas como resultado de necessidades não satisfeitas. As perguntas dos alunos oportunizam momentos de reflexão nos quais eles se sentem mentalmente envolvidos e desacomodados (CAMARGO et al., 2011), neste instante inicia-se um processo de inquietação, desencadeando a tomada de

consciência sobre os seus conhecimentos anteriores, de modo a fazer comparações e relações, no sentido de fomentar a organização da pergunta (CAMARGO et al., 2011). Esse conjunto de ações já se configura como momento de aprendizagem, pois ao apresentar sua pergunta, o sujeito está reestruturando o seu pensamento (SURMACZ; LOPES, 2018).

Apesar de importância do diálogo entre professor e o aluno, Barbosa, Fernandes e Campos (2011) observam que essa relação é deficiente mesmo durante as aulas presenciais ou em laboratório. Um dos grandes dificultadores do diálogo é a indisponibilidade do professor frente a multiplicidade de dúvidas e pela quantidade elevada de alunos. Essa deficiência torna-se ainda mais agravante nas atividades desenvolvidas com alunos iniciantes em programação onde altos níveis de reprovações e desistências são verificados.

No âmbito das tecnologias educacionais é percebida uma variedade de pesquisas e ferramentas que visam facilitar ou propiciar o diálogo com aluno ao longo de seus estudos. Os CAI (*Computer Aided Instruction*) e STIs (Sistemas Tutores Inteligentes) frequentemente buscam implementar mecanismos para simular o diálogo com o aluno e vem progressivamente alçando sucesso (ANDERSON; BOYLE; REISER, 1985; UENO, 1989; GERDES et al., 2017).

Já no contexto do ensino de programação as estratégias pedagógicas como a Programação em Pares (WILLIAMS et al., 2002), Aprendizado Baseado em Problemas (FEKETE; GREENING; KINGSTON, 1998), Revisão de Soluções por Não Autores (NOBRE; MENEZES, 2002), Competições de Programação, Formação de Grupos de Estudos (BORGES; FILHO, 2005) e as Arquiteturas Pedagógicas (CARVALHO; NEVADO; MENEZES, 2005), buscam, mesmo que indiretamente, uma maior promoção do diálogo durante as atividades desenvolvidas.

Carvalho, Nevado e Menezes (2005) propõem o conceito de arquiteturas pedagógicas, concebendo-as como micro ecossistemas cognitivos que englobam ideias epistemológicas relacionais, pedagogias abertas, tecnologias digitais e novos referenciais de tempo e espaço como condições estruturantes para as aprendizagens individuais e construções coletivas. As arquiteturas

encontram sustentação na articulação de uma concepção construtivista de aprendizagem e com a pedagogia da pergunta.

> Para cada pergunta há necessidade de se indexar situações que possam ajudar o estudante a iniciar e a prosseguir com um problema de um modo relativamente autônomo. Para ajudar na tarefa de resolução de problema pode-se apresentar, justapor e contrapor respostas e alternativas encontradas por especialistas, ou colegas, para problemas semelhantes. As tarefas implicam a definição de várias etapas, essas circunscrevem e identificam as características mais decisivas (ou etapas sem as quais não é possível avançar). (CARVALHO; NEVADO; MENEZES, 2005)

Para cada pergunta há necessidade de se indexar situações que possam ajudar o estudante a iniciar e a prosseguir com um problema de um modo relativamente autônomo. Para ajudar na tarefa de resolução de problema pode-se apresentar, justapor e contrapor respostas e alternativas encontradas por especialistas, ou colegas, para problemas semelhantes. As tarefas implicam a definição de várias etapas, essas circunscrevem e identificam as características mais decisivas ou etapas sem as quais não é possível avançar (CARVALHO; NEVADO; MENEZES, 2005).

O *feeback* quando relacionado a aprendizagem de programação é classificado por Keuning, Jeuring e Heeren (2018) da seguinte maneira:

- Conhecimento sobre restrições de tarefas;
- Conhecimento sobre conceitos;
- Conhecimento sobre erros;
- Conhecimento sobre como proceder;
- Conhecimento sobre metacognição.

A classificação criada por Keuning, Jeuring e Heeren (2018) levou em consideração 69 ambientes educacionais ligados a programação e 17 revisões recentes relacionadas ao processo de ensino dessa disciplina. Suas conclusões apontam a importância do atendimento ao aluno sobre diferentes perspectivas do conhecimento que se está desenvolvendo. Para Keuning, Jeuring e Heeren (2018) ainda é muito comum o tratamento de *feedbacks* associados aos erros no programa desenvolvido ou relacionados ao código-fonte, no entanto, os autores destacam a importância das demais variedades

de *feedbacks* a fim de oferecer as condições de desenvolvimento do aluno na aprendizagem da disciplina de programação.

3. Referencial Teórico

O objetivo desse capítulo é apresentar os conceitos considerados alicerces para o desenvolvimento desse trabalho.

Na Seção 3.1 serão apresentados os principais aspectos associados os agentes inteligentes e sistemas multiagentes.

Já na Seção 3.2 serão abordadas as técnicas computacionais ligadas à recuperação da informação e à classificação de documentos utilizando os modelos probabilísticos.

Por último, na Seção 3.3 serão tratados os aspectos relacionados aos padrões de projeto e de arquitetura de software.

3.1. Agentes Inteligentes e Sistemas Multiagentes

Nesta seção abordaremos os conceitos relacionados sistemas multiagentes, de modo individual trataremos na subseção 3.1.1 os agentes inteligentes e na subseção 3.1.2 abordaremos o uso de vários agentes inteligentes ligados a um mesmo sistema, cooperando ou competindo na solução de um problema.

3.1.1. Agentes Inteligentes

O termo agente racional ou inteligente é amplamente usado nas áreas de estudo da computação e é visto principalmente no campo de estudo da Inteligência Artificial. Segundo Russel e NORVIG (2013) uma definição rápida para agentes é que eles são entidades de software ou hardware capazes de reagir e tomar decisões de forma autônoma com base nas alterações do ambiente o qual estão situados.

Um agente é uma ferramenta que realiza alguma tarefa em nome de um humano. Por exemplo, um simples agente pode ser construído para comprar uma ação específica quando o preço dela cair abaixo de um determinado nível. Um simples agente de busca na Internet pode ser projetado para enviar consultas a uma série de ferramentas de busca e comparar os resultados (COPPIN, 2004). Para Russel e NORVIG (2013) um agente é tudo o que pode

ser considerado capaz de perceber seu ambiente por meio de sensores e de agir sobre esse ambiente por intermédio de seus dispositivos atuadores.

Um agente não é considerado inteligente se ele opera apenas com base no conhecimento anterior de seu projetista. Para ser uma agente inteligente é necessário que um agente tenha autonomia para agir segundo suas percepções (RUSSEL; NORVIG, 2013). Agentes inteligentes têm conhecimento adicional de domínio, o que os habilita a realizar as tarefas deles mesmo quando os parâmetros da tarefa mudam ou quando surgem situações inesperadas (COPPIN, 2004).

Segundo Russel e NORVIG (2013) ao compararmos os agentes inteligentes com os agentes humanos percebemos que os estes possuem sensores para a percepção do seu meio ambiente, tais como os ouvidos, olhos e outros órgãos, e também os elementos atuadores como as pernas, voz, braços, mãos e outras partes do corpo que permitem que os agentes humanos respondam os estímulos percebidos por seus sensores. De forma análoga, os agentes inteligentes computacionais são constituídos de sensores que permite a capturara de informações sobre o ambiente e seus atuadores permitem que estes agentes respondam ao ambiente onde estão inseridos. A Figura 2 ilustra esses mecanismos de um agente inteligente.

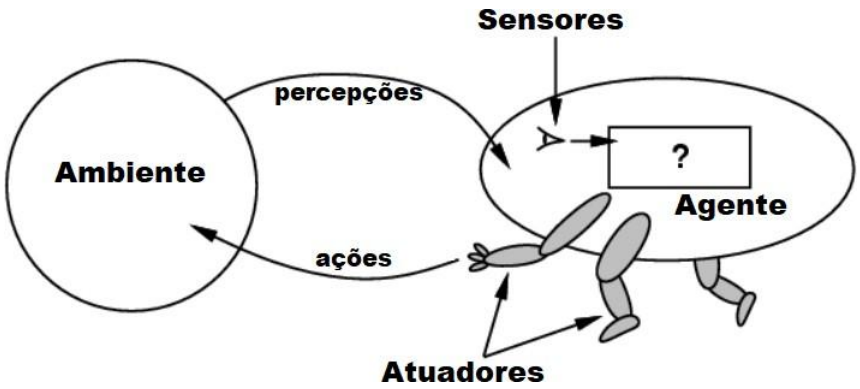


Figura 3 - Interação de agente com o ambiente por meio dos sensores e atuadores.

Fonte: (RUSSEL; NORVIG, 2013)

De acordo com Pascutti (2002), cada agente inteligente possui um conjunto de propriedades distintas que o apoia na obtenção de seus objetivos. No Quadro 1 são apresentadas essas propriedades catalogadas.

Quadro 1 - Propriedades de agentes inteligentes.

Propriedade	**Definição**
Interação	Um agente comunica-se com o ambiente e outros agentes.
Adaptação	Um agente adapta/modifica seu estado de acordo com as mensagens obtidas do ambiente.
Autonomia	Um agente é capaz de agir sem intervenção externa direta; ele possui seu próprio controle e pode decidir atender ou não uma mensagem.
Aprendizagem	Um agente pode aprender, com base em experiências anteriores enquanto reage e interage com seu ambiente.
Mobilidade	Um agente é capaz de mover-se de um ambiente para outro.
Colaboração	Um agente pode cooperar com outros agentes para alcançar seus objetivos ou do sistema.
Reatividade	Um agente é capaz de reagir a mudanças ocorridas no ambiente
Proatividade	Um agente é capaz de tomar iniciativas que visem alcançar seus objetivos.

FONTE: (PASCUTTI, 2002)

A tecnologia de agentes é defendida como um ativador principal para suportar uma nova era de crescente interação global e de uso de sistemas distribuídos. Dentre as vantagens no uso de agentes inteligentes, destaca-se a redução de custos, a melhora na eficiência e eficácia das aplicações, a maior flexibilidade para suportar os requisitos e o favorecimento a colaboração de indivíduos, grupos, empresas e universidades para trabalharem globalmente (ODELL, 2007).

3.1.1.1. Tipos de Agentes

Em Russel e NORVIG (2013) são definidos os tipos de agentes da seguinte forma:

- **Agente reativo:** Agente que escolhe sua ação baseado na percepção atual, ou seja, sem levarem consideração o histórico das percepções. Este é o tipo de agente mais simples. Sua estrutura básica resume numa condição, exemplo: SE situação= valor ENTÃO ação.
- **Agente reativo baseado em modelos:** Este tipo de agente mantém internamente um modelo do seu ambiente e, para selecionar uma ação, leva em consideração o histórico das percepções.
- **Agente baseado em objetivos:** Este tipo de agente pondera suas ações sempre levando em consideração a tentativa de alcançar seus objetivos, assim ele combina seus objetivos com o modelo (as mesmas informações que foram usadas no agente reativo baseado em modelo), a fim de escolher ações que esteja, mas adequadas aos objetivos estabelecidos.
- **Agente baseados em utilidades:** Este tipo de agente usa um modelo do mundo juntamente com uma função utilidade que mede suas preferências entre estados do mundo. Em seguida, ele escolhe a ação que leva à melhor utilidade esperada, na qual a utilidade esperada é calculada pela média entre todos os estados resultantes possíveis, ponderados pela probabilidade do resultado.
- **Agentes com aprendizagem:** Este é um tipo de agente capaz de aprender. Sua vantagem está na capacidade de operar em ambientes inicialmente desconhecidos e se tornar mais competente do que seu conhecimento inicial sozinho poderia permitir.

3.1.1.2. Tipos de Ambientes

Uma arquitetura básica de um agente inteligente deve assegurar que ele tenha o devido contato seu ambiente de atuação, além disso cada agente possui internamente os mecanismos de inferência que o permite avaliar as informações recebidas de seus sensores e também agir a partir de seus

mecanismos atuadores. E então, uma base de conhecimento é utilizada pelos agentes para armazenar seus conhecimentos adquiridos.

Para Russel e NORVIG (2013) é primordial identificar o tipo de ambiente onde um agente atuará, para apenas depois decidir o tipo de agente e a arquitetura onde ele será implementado. O local de atuação de um agente é conhecido como "ambientes de tarefas", que são essencialmente os “problemas” para os quais os agentes irão dá “soluções”. Russel e NORVIG (2013) classificam os ambientes com sendo: completamente ou parcialmente observável, de participação de um único ou multiagentes, episódico versus sequencial, estático versus dinâmico, conhecido versus desconhecido, quando diz respeito ao estado de conhecimento do agente (ou do projetista) sobre as “leis da física” no meio ambiente, ou seja, se é sabido toda causas e efeitos neste

3.1.2. Sistemas Multiagentes

A partir do entendimento da constituição de agente inteligente e sua atuação num ambiente, podemos pensar numa comunidade de agentes competindo ou cooperando na resolução de um problema. De acordo com Signoretti (2012) um sistema multiagente (SMA) é um sistema que possui vários agentes, cada um com sua função, interagindo entre si por um objetivo.

Os sistemas multiagentes são indicados para a resolução de problemas complexos e permite maior flexibilidade ao distribuir recursos computacionais (JUNIOR et al., 2003). Por exemplo, ao invés de aumentar o número de processadores ou a capacidade de memória, múltiplos agentes num sistema

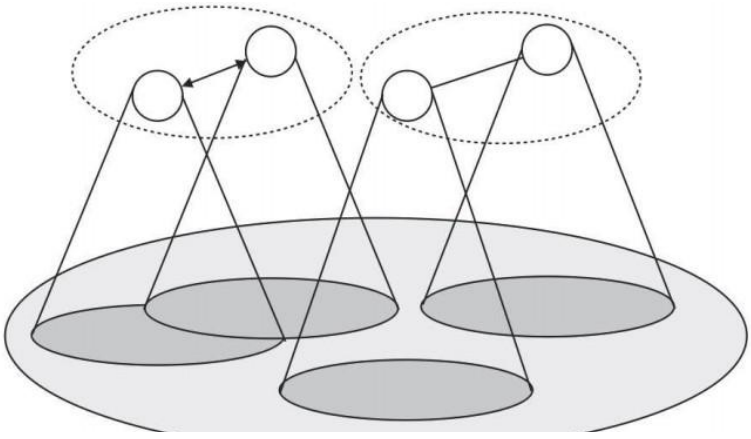


Figura 4 - Estrutura típica de um sistema multiagente.

Fonte: Adaptado de (WOOLDRIDGE, 2009).

distribuído poderiam colaborar na solução de um problema.

Por outro lado, como ilustra a Figura 3, múltiplos agentes podem operar em diferentes perspectivas do ambiente e juntos cooperarem em partes da solução de um problema maior. Na parte superior da Figura 3 ocorre as instâncias dos agentes inteligentes e a comunicação entre si. Na parte inferior da Figura 3 é ilustrado o ambiente que o sistema multiagente atua, enquanto que a parte do meio, na cor cinza, da Figura 3 ilustra as esferas de influência de cada agente sobre o ambiente. A construção de sistemas multiagentes é apoiada por diversas ferramentas e nas mais variadas linguagens de programação.

3.2. Recuperação da Informação

A Recuperação da Informação (RI) é uma área originária da Ciência da Computação que visa facilitar o acesso dos usuários a informações relevantes as suas necessidades. Entretanto, essa não é uma tarefa trivial devido ao grande volume de dados disponíveis nos atuais sistemas de informação (BAEZA-YATES; RIBEIRO-NETO, 2013).

Diversas abordagens têm sido empregadas na recuperação de informação. No que se refere a mineração de dados, esta objetiva consultar e fornecer os dados da forma mais adequada, enquanto que outras abordagens são dedicadas a entender a consulta do usuário, de forma que os dados retornados correspondam adequadamente aos critérios definidos na consulta (GONZALEZ; LIMA, 2003).

> O termo *Recuperação da informação* pode ainda ser empregado para designar a operação que fornece uma resposta mais ou menos elaborada a uma demanda, e esta resposta é convertida num produto cujo formato é acordado com o usuário (bibliografia, nota de síntese e etc.). Há ainda autores que conceituam a recuperação de informação de forma muito mais ampla, ao subordinar à mesma o tratamento da informação, (catalogação, indexação, classificação). (FERNEDA, 2003)

Segundo Ferneda (2003), diferentemente da recuperação da informação em banco de dados, onde uma consulta resulta precisamente às condições formuladas através de uma expressão de busca, em um sistema de

recuperação de informação essa precisão não é tão estrita. Os sistemas de recuperação de informação lidam com objetos linguísticos, dados desestruturados e em muitos casos herdam toda a problemática inerente ao tratamento da linguagem natural.

Segundo Ferneda (2003), diferentemente da recuperação da informação em banco de dados, onde uma consulta resulta precisamente às condições formuladas através de uma expressão de busca, em um sistema de recuperação de informação essa precisão não é tão estrita. Os sistemas de recuperação de informação lidam com objetos linguísticos, dados desestruturados e em muitos casos herdam toda a problemática inerente ao tratamento da linguagem natural.

A indexação da informação é a principal função de Sistema de Recuperação da Informação (SRI), sendo necessária para adequadamente representar, armazenar, organizar e localizar os itens de informação (FERNEDA, 2013). Dentre os componentes de um SRI, destacam-se os documentos (ou corpus), a representação, as necessidades dos usuários, a consulta formulada e a função de busca propriamente dita. Uma representação simplificada do processo de recuperação de informação é apresentada na Figura 4.

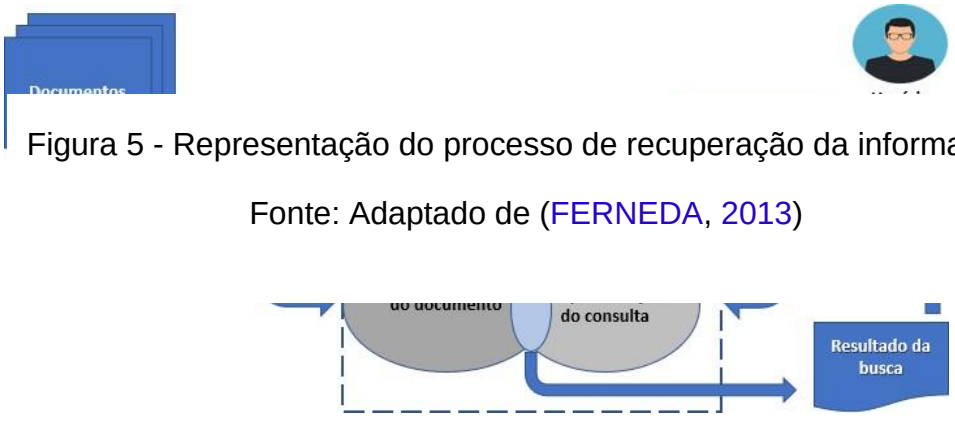


Figura 5 - Representação do processo de recuperação da informação.

Fonte: Adaptado de (FERNEDA, 2013)

Os documentos (Figura 4) são entendidos como coisa informativa, incluindo objetos, artefatos, imagens, sons etc. Baeza-Yates e Ribeiro-Neto (2013) classificam os documentos como texto, links e multimídia. Os documentos de textos são dados definitivamente desestruturados, os links são

documentos relacionados a web, os textos semiestruturados estão relacionados aos documentos baseados em XML e por último os documentos de multimídia estão ligados a sons e imagens.

A representação de um documento (Figura 4), por outro lado, tem por objetivo identificar e descrever resumidamente o seu conteúdo informacional, ao mesmo tempo em que define seus pontos de acesso usados pela função de busca num SRI.

A classificação de documentos pode ser realizada a partir da Engenharia de Conhecimento, onde através de um trabalho manual o engenheiro do conhecimento com o auxílio de um especialista no domínio da aplicação realiza a catalogação. Esse trabalho resulta num conjunto de regras que serão utilizadas para representar o conhecimento e também para representar a consulta nos termos equivalentes a classificação realizada (RODRIGUES, 2009).

Outra alternativa empregada no processo de classificação de documentos é a Aprendizagem de Máquina, nesta abordagem são criados algoritmos capazes de classificar documentos não vistos antes com base num conjunto de documentos previamente classificados (BAEZA-YATES; RIBEIRO-NETO, 2013).

A representação da consulta (Figura 4) refere-se a tarefa de modelar a consulta dos usuários, por um lado, sob a perspectiva de sistemas centrados no usuário, busca-se facilitar que o usuário expresse a consulta ao seu modo, sem que ele precise de todos os termos usados na catalogação do documento, normalmente essa consulta é feita através de linguagem natural.

> Para que seja possível uma comparação entre a expressão de busca e cada um dos documentos do corpus é necessário que essa seja representada de forma similar à representação dos documentos. Diversos recursos podem ser oferecidos por um sistema a fim de facilitar o usuário na especificação de sua expressão de busca. Porém, essa consulta deve ser representada de uma forma idêntica ou similar à utilizada pelos documentos. (FERNEDA, 2013).

Por último, e mais importante, o processo de recuperação da informação possui o elemento Função de Busca (Figura 4) que é responsável por receber

a consulta do usuário devidamente representada e então realiza a busca dos documentos que atendam aos termos da consulta.

De acordo com Catae (2012) e Ferneda (2013), de forma geral, a função de busca calcula o grau de similaridade entre a expressão de busca e cada um dos documentos do corpus. Já para Baeza-Yates e Ribeiro-Neto (2013) as abordagens computacionais podem ser categorizadas de acordo com as propriedades do documento. A Figura 5 ilustra essa taxonomia ao apontar os tipos de documentos e as abordagens utilizadas para modelar os documentos e sua consulta.

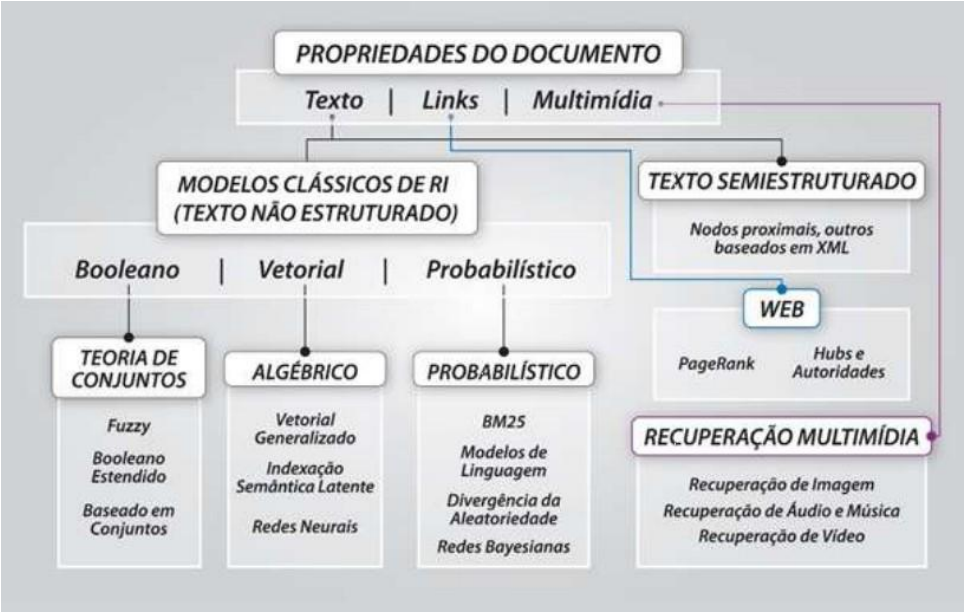


Figura 6 - Taxonomia do documento.
Fonte: (RIBEIRO-NETO e BAEZA-YATES, 2013)

A discussão da cada abordagem ilustrada na Figura 5 demandaria um trabalho específico somente com esse foco. Abordaremos nesta pesquisa os modelos vetoriais e os probabilísticos empregados no tratamento de textos desestruturados. As Subseções seguintes deste capítulo adentrarão nesta discussão.

3.2.1. Classificação de Textos

A classificação de texto além de ligado a classificação de documentos também é uma das tarefas fundamentais no processamento de linguagem natural,

especificamente na NLU (Natural Language Processing ou Processamento de Linguagem Natural) por permitir a criação de rótulos específicos para um dado texto (YANG et al., 2016; CATAE,2012). Tal recurso tem amplas aplicações, incluindo rotulagem de tópicos (LUCCA et al., 2013; RODRIGUES, 2009), classificação de sentimentos (OGURI; LUIZ; RENTERIA, 2006), a detecção de spam YANG et al. e no rotulamento de intenções contidas nos turnos de uma conversação (RATO, 2016).

> *Durante a indexação são extraídos conceitos do documento através da análise de seu conteúdo o e traduzidos em termos de uma linguagem de indexação, tais como cabeçalhos de assunto, tesauros, etc. classificação single-label (e.g., uma mensagem classificada exclusivamente como spam ou não-spam). Quando um documento pode ser associado a mais de uma categoria ao mesmo tempo, dizemos que como classificação multi-label (e.g., notícias classificadas como pertencentes a mais de um assunto) (FERNEDA, 2003).*

São diversas as abordagens computacionais empregadas no reconhecimento de textos, muitas dessas técnicas são provenientes do PLN/NLU e empregam a análise morfológica, semântica e sintática do documento. No entanto, Rato (2016) argumenta que identificar morfologicamente as palavras utilizadas num diálogo não é mais importante que identificar qual a intenção contida nesta mensagem, dado que um diálogo segue um ciclo, conhecido como o ciclo de diálogo, onde sempre que um dos participantes diz alguma coisa, o conteúdo é interpretado pelo ouvinte, que então toma uma decisão sobre como responder.

Devido ao formato desestruturado dos documentos textuais e sua característica incerta, diversas técnicas baseadas no raciocínio probabilístico vêm sendo empregado na NLU (BAEZA-YATES; RIBEIRO-NETO, 2013).

Representação Vetorial de Textos

3.2.2. Representação Vetorial de Textos

Para Baeza-Yates e Ribeiro-Neto (2013) a representação do texto com base no modelo booleano é limitada, tendo em vista que neste modelo é apenas indicado se um dado termo do vocabulário está presente ou não no documento. O modelo vetorial é uma alternativa que visa representar não somente a presença do termo no documento, mas sua frequência ou peso no documento

(BAEZA-YATES; RIBEIRO-NETO, 2013). Essa abordagem, segundo os autores, permite que um grau de similaridade seja estabelecido entre a consulta e os documentos representados vetorialmente.

Uma representação vetorial com base na frequência de termos (*tf*) nos documentos é muito utilizada em cálculos de similaridade. O *tf* (termo frequência), também conhecido como frequência absoluta (ANDRADE, 2015), indica a quantidade de ocorrências de um dado termo *t* em um determinado documento *d,* como demostrado a seguir:

$$Tf_{t,d} = Freq_{t,d}$$

De acordo com Andrade (2015) esta pode ser considerada uma medida simples, porém apresenta desvantagens. A primeira refere-se ao fato desse modelo não levar em conta a quantidade de palavras existentes nos documentos. Com isso, um termo que ocorre poucas vezes em um documento pequeno poderá ter a mesma frequência de um termo que ocorre muitas vezes em documentos com grande quantidade de palavras. Outra desvantagem é o fato de não ser possível identificar se um termo ocorre em poucos ou em muitos documentos.

Uma outra abordagem representacional básica na estruturação de documentos textuais é o *Bag of Words* (BOW) (SOARES, 2017), consiste de uma representação vetorial onde cada termo (palavra) é uma dimensão do espaço a ser considerado. Uma coleção de documentos é normalmente representada por uma matriz de documentos-termos, onde cada linha representa um documento (vetor de documento) da coleção e cada coluna representa um termo (vetor de termo). As células da matriz são preenchidas, normalmente, com a frequência (*tf*) em que os termos aparecem nos documentos.

Finalmente, técnica chamada “inverse document frequency” (*idf*) considera a inversão da frequência de um termo, minimizando sua relevância à medida que ele aparece em muitos documentos do corpus. Logo, se todos os documentos do corpus contiverem um determinado termo, o idf desse termo

será igual a um (FERNEDA, 2013). O *idf* de um termo é calculado da seguinte forma:

$$idf_t = \frac{N}{n_t}$$

Onde **N** é o número de documentos no corpus e n_t é o número de documentos que contém o termo *t*.

O produto $tf * idf$ é utilizado para atribuir peso a cada elemento dos vetores que representam os documentos do corpus. De acordo com Ferneda (2013) os melhores termos de indexação, os que apresentarão maior peso, são aqueles que ocorrem com uma grande frequência em poucos documentos.

3.2.3. O Algoritmo *Naive Bayes*

O *Naive Bayes* é um algoritmo probabilístico baseado na regra de Bayes, no qual estimativas são estabelecidas com base na probabilidade condicional P (causa|efeito) partir das ocorrências anteriores destes mesmos eventos (RUSSEL; NORVIG, 2013).

$$P(\frac{causa}{efeito}) = \frac{P(\frac{causa}{efeito}) * P(causa)}{P(efeito)}$$

A regra de Bayes é definida da seguinte forma:

No *Naive Bayes* se for dado um conjunto de j classes $C = c1, c2, ..., c_j$ e outro de i documentos $D = d1, d2, ..., d_i$ descritos por um espaço multidimensional composto pelas palavras ou termos que aparecem em toda a coleção, o algoritmo aprende uma função de classificação $f: X \rightarrow C$ que mapeia os documentos nas classes (OGURI; LUIZ; RENTERIA, 2006; LUCCA et al., 2013).

A aplicação do *Naive Bayes* na classificação de textos é devido à sua eficiência e capacidade de lidar com grande quantidade de dados (CATAE, 2012). Embora sua característica ingênua (ANDRADE, 2015), por não

considerar dependência entre as ocorrências das propriedades ou na ordem que elas aparecem no texto, o que raramente é observado em problemas reais (FERNEDA, 2013), mesmo assim o *Naive Bayes* tem oferecido ótimos resultados. Ao aplicar o *Naive Bayes* na tarefa de classificação de texto pressupõe-se que as distribuições das palavras num documento são geradas por um modelo paramétrico específico, então esses parâmetros poderão ser estimados a partir dos dados de treinamento previamente classificados (SU; SHIRAB; MATWIN, 2011).

Existem diversas variações de algoritmos *Naive Bayes*, fundamentalmente destacam-se os seguintes: o modelo Bernoulli e o modelo multinomial, também conhecido como *Multinomial Naive Bayes* (MNB).

No modelo Bernoulli também chamado de modelo binário (ANDRADE,2015), cada atributo pode indicar ou não a ocorrência de um evento no documento. O modelo binário tem sido muito empregado na classificação de sentimentos positivos e negativos expressados em textos (SHIMODAIRA, 2014; FERNEDA, 2003), identificação de spams e dentre

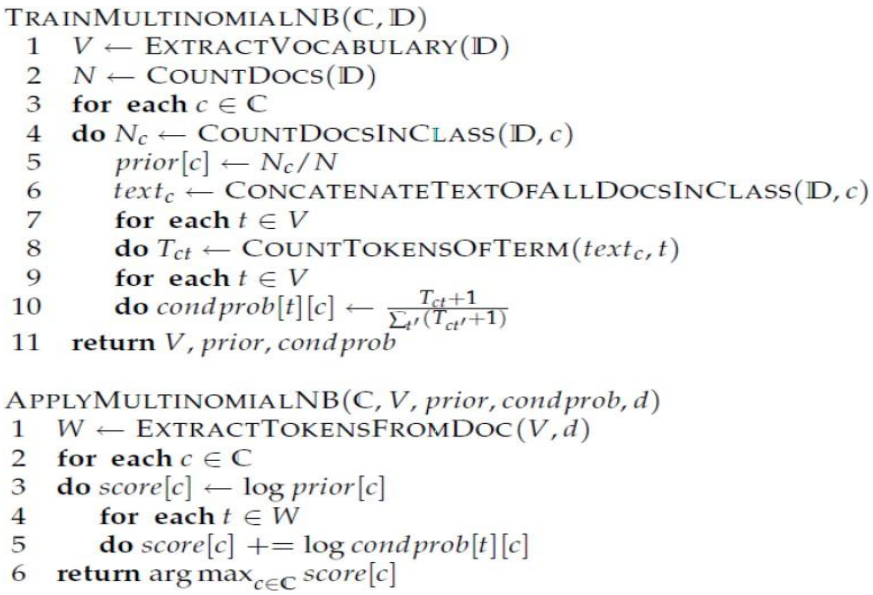


Figura 7 - Pseudocódigo do algoritmo *Naive Bayes*.

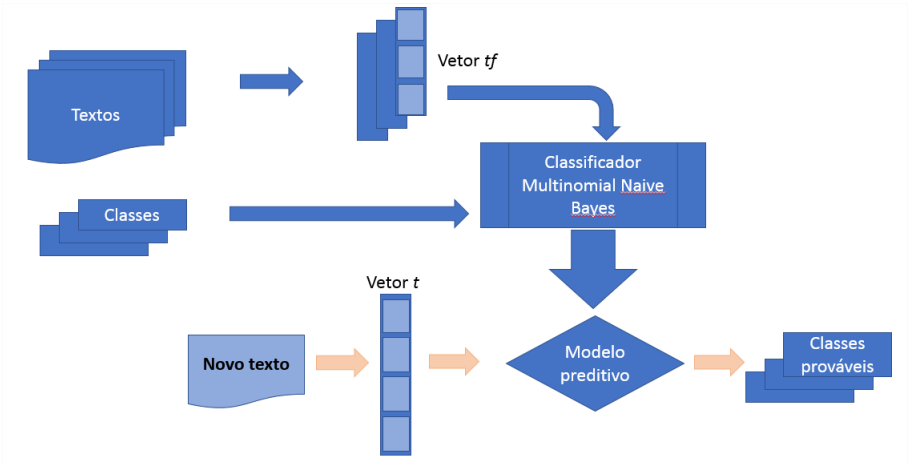
Fonte: (MANNING; RAGHAVAN; SCHÜTZE,2008)

outros cenários de classificação binária.

Um classificador de texto MNB recebe como parâmetros, além do documento de teste, o conjunto de classes, o vocabulário e as probabilidades estimadas no treinamento (LUCCA et al., 2013). Para cada classe, a probabilidade a posteriori é calculada somando o logaritmo da probabilidade a priori com os logaritmos das probabilidades condicionais de cada termo do documento de teste. O documento então é rotulado com a classe que obtiver a maior probabilidade a posteriori. A Figura 6 ilustra o pseudocódigo do MNB.

Devido a capacidade de trabalhar com o número de ocorrência, na classificação de textos é realizada a criação de um modelo vetorial representando os termos encontrados num texto e suas respectivas frequências. Ao ser treinado com uma base classificada a priori, o MNB cria um modelo capaz de prevê a ocorrência dessas classes a partir de um novo texto não classificado. A Figura 7 ilustra o processo de classificação aplicado por (LUCCA et al., 2013) utilizando o MNB e um vetor de frequência termos.

A maior vantagem do MNB é sua escalabilidade (PUURULA, 2012), sua simples implementação (SU; SHIRAB; MATWIN, 2011)) e alta eficiência (RODRIGUES, 2008). Todavia, mesmo com suas diversas vantagens, o MNB falha em documentos onde não a presença de termos usados na classe, na identificação da probabilidade de uma classe envolve tomar um produto de probabilidades de cada termo, logo se qualquer um dos termos do produto for zero, então o produto inteiro será zero (PUURULA, 2012), o que nem sempre é



verdade.

Figura 8 - Fluxograma de classificação de texto com MNB.
Fonte: Autoria Própria.

Uma solução de contorno para o problema da probabilidade zero é inicializar o peso de cada termo com o valor 1 em vez de 0, essa técnica é chamada de correção Laplace. Todavia, a correção Laplace tende a influenciar a predição a favor das classes majoritárias, uma vez que os termos irrelevantes poderão obter peso 1 e logo será processado na predição fazendo com que seja considerado no processamento. O Laplace também desnormalizará a distribuição de probabilidades do modelo, tornando-o com muitos termos com probabilidade 1, logo implicará num processamento desnecessário para as ocorrências únicas de um dado termo (RODRIGUES, 2009). Alternativas como a normalização por classes (MNB_{pcn}) visam evitar este problema (FRANK; BOUCKAERT, 2006).

3.3. Padrões de Software

O padrão de um software é uma descrição do problema e da essência de sua solução, de modo que a solução seja descrita de forma que possa ser reutilizada noutros problemas semelhantes (SOMMERVILLE, 2011) e aplicável a diferentes contextos do projeto de software (BOOCH; RUMBAUGH; JACOBSON, 2012).

Pressman (2011) define que um projeto de software baseado em padrões cria uma nova aplicação através da busca de um conjunto de soluções comprovadas para um conjunto de problemas claramente delineados. Para isto, cada solução proposta para o tratamento de um dado problema é descrito por um padrão de software que foi anteriormente investigado em sua essência e devidamente catalogado por outros engenheiros de software que se depararam com o problema, implementaram uma solução, sendo esta reutilizada e aplicada variadas vezes noutras aplicações.

Os padrões de projeto abrangem um amplo espectro de abstração e aplicação, em seguida será apresentado as principais categorias desses padrões:

- **Padrões de análise:** apresentam soluções para problema de análise de sistemas, que são encontrados em um domínio específico;
- **Padrões de arquitetura** ou *Architectural Patterns*: apresentam o esquema ou organização estrutural fundamental de sistemas de software;
- **Padrões de projeto** ou *Design Patterns*: refina os componentes de uma arquitetura de software e descrevem soluções para problemas de projeto de software. Esses padrões fornecem uma especificação explicita da interação de classes e objetos, melhorando a documentação e a manutenção de sistemas;
- **Padrões de programação**: descrevem boas práticas de uma determinada linguagem ou regras gerais de estilo de programação da solução.

Nas subseções seguintes será abordado o padrão arquitetural aplicado na construção de sistemas web e os principais *Design Patterns*.

3.3.1. Padrões Arquiteturais

Existem padrões arquiteturais específicos para cada tipo de software: sistemas distribuídos, adaptativos e interativos dentre outros.

O resultado do processo de projeto de arquitetura é um modelo de arquitetura que descreve como o sistema está organizado num conjunto de componentes de comunicação. Pressman (2011) destaca que um projeto de arquitetura é composto ao menos pelas seguintes propriedades:

- **Propriedades estruturais:** Esse aspecto do projeto define os componentes do sistema (como módulos, objetos, filtros e camadas) e a maneira pela qual esses componentes são integrados e comunicam entre si;
- **Propriedades não funcionais:** Definem a maneira pela qual o projeto de arquitetura atingirá os requisitos de desempenho,

capacidade, confiabilidade, segurança, adaptabilidade e outras características do sistema que não sejam aquelas funcionalidades acessadas diretamente pelo usuário;

- **Famílias de sistemas**: Descrevem os padrões utilizados para que o projeto tenha as condições de ser reutilizado ou de se aproveitar a reutilização de outros componentes.

3.3.1.1. A arquitetura em Camadas

A abordagem em camadas permite que o software possa ser desenvolvido em partes com diferentes responsabilidades onde cada camada é desenvolvida para prestar um conjunto de serviços, à medida que também utiliza os serviços de camadas inferiores (GAMMA, 2011). A grande vantagem do modelo em camadas é seu baixo acoplamento entre as partes do sistema, sua mutabilidade e portabilidade, pois enquanto a interface de uma camada não for alterada os seus mecanismos internos poderão ser substituídos ou melhorados (GAMMA, 2011). Além disso, quando a interface da camada muda ou tem novos recursos adicionados, apenas a camada adjacente é afetada (SOMMERVILLE, 2011).

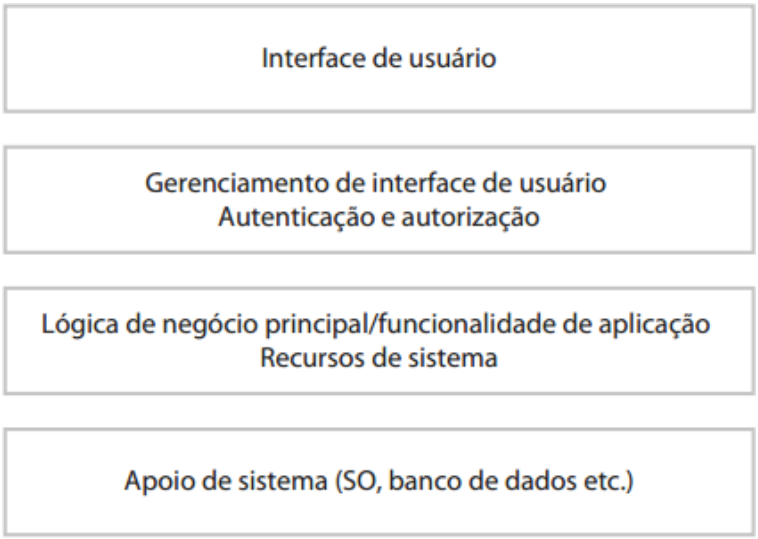


Figura 9 - Exemplo de Arquitetura em 4 Camadas.
Fonte: (SOMMERVILLE, 2011)

No exemplo da Figura 8, a camada mais baixa inclui o software de apoio ao sistema, como por exemplo, os sistemas de banco de dados e o sistema operacional (SOMMERVILLE, 2011).

A próxima camada é a camada de aplicação, que inclui os componentes relacionados com a funcionalidade do sistema e os componentes utilitários que serão utilizados por outros módulos da aplicação. A terceira camada está dedicada ao gerenciamento de interface de usuário e ao fornecimento de autenticação, enquanto que a quarta camada é responsável por apresentar os recursos da interface com o usuário. Os números de camadas não são arbitrários, podendo o mesmo exemplo da Figura 8 ser representado com mais ou menos camada (SOMMERVILLE, 2011).

Na Figura 9 é ilustrada uma arquitetura web em 3 camadas utilizada num sistema de Internet Banking. Neste sistema a camada 1 é responsável por disponibilizar a interface com o usuário e as funcionalidades acessadas por ele. No caso da Web, a aplicação é acessada através Browser, a partir do protocolo HTTP utilizado para troca de mensagens como servidor Web responsável por acomodar camada 2 da aplicação.

O servidor Web fornece os serviços de gerenciamento de dados, como por exemplo, a geração de página Web e alguns serviços da aplicação. No

exemplo da Figura 9 os serviços de aplicação, tais como os recursos para transferir o dinheiro, gerar os extratos, pagar as contas, e assim por diante, são implementados no servidor Web a partir de scripts, os quais são executados pelo cliente.

Por último, na camada 3 é disponibilizado o servidor de banco de dados, responsável por persistir os dados e garantir seu gerenciamento.

Este sistema em camadas, como exemplificado na Figura 9, é escalável, pois é relativamente fácil adicionar novos servidores (escalabilidade para cima), assim como aumentar o número de clientes.

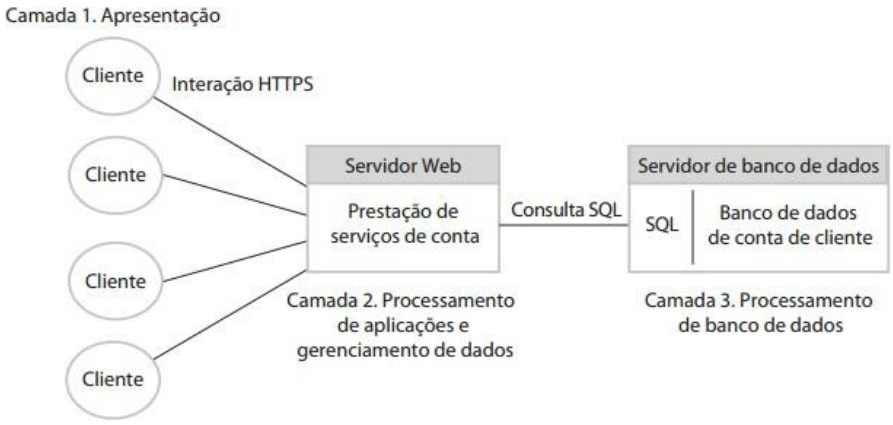


Figura 10 - Arquitetura de três camadas para um sistema de Internet banking.
Fonte: (SOMMERVILLE, 2011)

O modelo cliente-servidor de três camadas pode ser estendido para uma variante em multicamadas, na qual os servidores adicionais são adicionados ao sistema. Esse processo envolve o uso de um servidor Web para gerenciamento de dados e de servidores separados para processamento de aplicação e serviços de banco de dados.

O modelo cliente-servidor de três camadas pode ser estendido para uma variante em multicamadas, na qual os servidores adicionais são adicionados ao sistema. Esse processo envolve o uso de um servidor Web para gerenciamento de dados e de servidores separados para processamento de aplicação e serviços de banco de dados.

O modelo MVC é um padrão arquitetural muito empregado em sistemas Web (ROSSI et al, 2016), esse padrão foi originalmente proposto na década de

1980 como uma abordagem de projeto de GUI que permitia várias formas de apresentação de um mesmo objeto e a utilização de diferentes estilos nas interações essas interfaces (SOMMERVILLE, 2011).

O MVC é composto pela tríade das camadas Modelo, Visão e Controlador (Model-View-Controller), conforme ilustra a Figura 10. O Modelo (Model) contém os dados e as operações sobre ele (a regras no negócio); a Visão (View) gerencia informações a serem mostradas ao usuário; e o Controlador (Controller) que processa os eventos do usuário (por exemplo, teclas, cliques do mouse etc.) e então repassa essas interações para o Modelo e a Visão (SOMMERVILLE, 2011; NAKAGAWA, 2006).

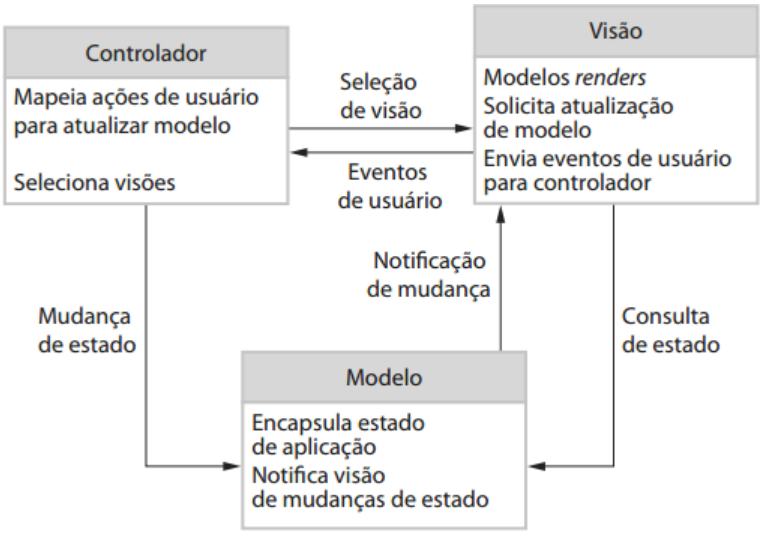


Figura 11 - Comunicação no modelo MVC.
Fonte: (SOMMERVILLE, 2011)

Lemos et al. (2013) argumentam que embora as aparências, é necessário não confundir modelo em 3 camadas tradicional com o modelo MVC. Enquanto que na arquitetura em 3 camadas tradicional a camada de apresentação nunca se comunica diretamente com a camada de dados (o Modelo), diferentemente, na arquitetura MVC, conforme ilustrado na 10, a comunicação entre as camadas ocorre de forma triangular: a View envia uma modificação para o Controller; após receber esse sinal, o Controller atualiza o Model, sendo a View atualizada diretamente a partir do Model ou do Controller.

3.3.1.2. Arquitetura Orientada a Serviços

A arquitetura orientada a serviços (SOA, do inglês *Service Oriented Architectures*) tem como essência o fornecimento de recursos independentes das aplicações que os usa. SOA é uma forma de desenvolvimento de sistemas distribuídos onde os componentes de sistema são serviços autônomos, executando em computadores geograficamente distribuídos (GUINARD et al., 2010; SOMMERVILLE, 2011).

Os provedores de serviços podem desenvolver serviços especializados e oferecê-los para uma variedade de usuários de serviço de diferentes organizações. De acordo com (SOMMERVILLE, 2011) uma distinção fundamental entre um serviço e um componente de software é que os serviços devem ser independentes e fracamente acoplados, ou seja, eles sempre devem operar da mesma maneira, independentemente de seu ambiente de execução. Pelo fato dos SOAs serem uma arquitetura menos rígidas, nas quais as ligações de serviços podem mudar durante a execução, uma versão diferente, mas equivalente, de um serviço, pode ser executada em diferentes momentos, permitindo maior flexibilidade na evolução e manutenção de recursos de uma solução.

No contexto da combinação de aplicações distribuídas e Web, as arquiteturas orientadas a serviços são conhecidas como Web Services (GUINARD et al., 2010; MCILRAITH; SON; ZENG, 2001), tal abordagem arquitetural tem transformado a web de apenas um repositório de textos e imagens para um provedor de recursos que garante integrações entre as mais variadas aplicações (MCILRAITH; SON; ZENG, 2001; GOTTSCHALK et al., 2002).

3.3.2. Padrões de Projeto

Um padrão de projeto nomeia, abstrai e identifica os aspectos-chave de uma estrutura de projeto comum para torná-lo para a criação de um projeto orientado a objetos reutilizáveis (GAMMA, 2011).

O padrão de projeto identifica as classes, instâncias participantes, seus papéis, as colaborações e a distribuição das responsabilidades. Cada padrão de projeto focaliza um problema ou tópico particular de projeto orientado a

objetos. Cada padrão descreve em que situação ele pode ser aplicado, se ele pode ser utilizado em função de outras restrições de projeto e as consequências, custos e benefícios de sua utilização.

Sommerville (2011) elenca quatro elementos essenciais nos padrões de projeto definidos pela "Gangue dos Quatro", em seu livro de padrões de projeto, são eles:

1. Um nome que seja uma referência significativa para o padrão;
2. Uma descrição da área de problema que explique quando o modelo pode ser aplicado;
3. A descrição da solução das partes da solução de projeto, seus relacionamentos e suas responsabilidades;
4. Uma declaração das consequências — os resultados e compromissos — da aplicação do padrão.

Gamma et al (2007) classificam os padrões de projeto em dois critérios, conforme apontado no Quadro 2. O primeiro critério, chamado de Propósito, reflete a finalidade de um padrão, ou seja, é aquilo que ele almeja fazer. Já o segundo critério refere-se ao Escopo onde é especificado o padrão que será aplicado, se sob as classes ou sob os objetos.

Quadro 2 - Catálogo de Design Patterns.

		Propósito		
		De criação	**Estrutura**	**Comportamental**
Escopo	**Classe**	Factory Method	Adapter (class)	Interpreter Template Method
	Objeto	Abstract Factory Builder Prototype Singleton	Adapter (object) Bridge Composit e Decorator Façade Flyweight Proxy	Chain of Responsibility Command Iterator Mediator Memento Observer State Strategy Visitor

Fonte: Adaptado de (GAMMA, 2011)

Gamma et al (2011) classificam os padrões de projeto em dois critérios, conforme apontado no Quadro 2. O primeiro critério, chamado de Propósito, reflete a finalidade de um padrão, ou seja, é aquilo que ele almeja fazer. Já o segundo critério refere-se ao Escopo onde é especificado o padrão que será aplicado, se sob as classes ou sob os objetos.

Sob a perspectiva comportamental de um padrão, Gamma (2011) descrevem da seguinte forma:

- **Padrões de Criação:** os padrões de criação abstraem o processo de instanciação. Eles ajudam a tornar um sistema independentemente de como seus objetos são criados, compostos e representados. Os padrões de criação se tornam importantes à medida que os sistemas evoluem no sentido de depender mais da composição de objetos do que da herança de classes. Quando isso acontece, a ênfase se desloca da codificação rígida de um conjunto fixo de comportamentos para a definição de um conjunto menor de comportamentos fundamentais.
- **Padrões Estruturais:** os padrões estruturais se preocupam com a forma como classes e objetos são compostos para formar estruturas maiores. Os padrões estruturais de classes utilizam a herança para compor interfaces ou implementações. Esse padrão é particularmente útil para fazer bibliotecas de classes desenvolvidas independentemente trabalharem juntas. Destaca-se neste padrão o Adapter e o Façade. O padrão Adapter é usado para converter a interface de uma classe em outra interface, esperada pelos clientes. O Façade define uma interface de nível mais alto que torna o subsistema mais fácil de ser usado.
- **Padrões Comportamentais:** os padrões comportamentais se preocupam com algoritmos e a atribuição de responsabilidades entre objetos. Os padrões comportamentais não descrevem apenas padrões de objetos ou classes, mas também os padrões de comunicação entre eles. Esses padrões caracterizam fluxos de controle difíceis de seguir em tempo de execução. Eles afastam o

foco do fluxo de controle para permitir que você se concentre somente na maneira como os objetos são interconectados.

3.4. Considerações Finais sobre o Capítulo

Os padrões de software se apresentam como de fundamental importância quando se deseja alinhar os requisitos funcionais e não funcionais com boas práticas já estabelecidas e testadas por outros engenheiros de software e além do mais, eles permitem que soluções exaustivamente testadas sejam aplicadas num novo projeto propiciando qualidade e ganho de tempo na especificação de uma solução.

A aplicação dos padrões arquiteturais permite definir estruturas robustas com alto poder de integração entre componentes, camadas e sistemas, de forma a manter separada a responsabilidade de um recurso e ter um baixo acoplamento entre eles, mas garantido a comunicação necessária entre as diferentes partes que constitui um sistema.

Os padrões de projeto também visam garantir que as classes e objetos atendam devidamente aos seus propósitos e escopos, estabelecendo-os nas melhores práticas e abordagens de implementação e uso com o objetivo de se obter o máximo proveito na utilização da programação orientada a objetos.

Para o escopo desse trabalho será utilizado a arquitetura MVC e a arquitetura SOA por permitir que aplicações sejam criadas com baixo acoplamento e uma boa distribuição de responsabilidades estre camadas do sistema.

Os Design Patterns *Façade*, *Proxy* e *Mediator* também serão aplicados na construção do modelo conceitual. O *Façade* por possibilitar agregar um conjunto de atributos e funcionalidades a partir de uma interface única, o padrão *Proxy* por permitir intercambiar a comunicação com recursos externos, tais como os serviços disponibilizados via SOA e o padrão *Mediator* por possibilitar a propagação de mensagens a diversos outros objetos conectados.

4. Trabalhos Correlatos

O levantamento dos trabalhos correlatos consistiu de uma pesquisa bibliográfica nas publicações realizadas no período de 2008 a 2018, assim foram consideradas as bases brasileiras WCBIE (Anais dos Workshops do Congresso Brasileiro de Informática na Educação), RBIE (Revista Brasileira de Informação na Educação), SBIE (Simpósio Brasileiro de Informação na Educação), WIE (Workshop de Informática na Escola) e também as publicações internacionais disponíveis no repositório IEEE.

As buscas nos trabalhos nacionais foram realizadas através das palavras chaves contidas nos títulos ou nos resumos, de acordo com a seguinte *string* de consulta:

> (portal ***OR*** software ***OR*** sistema ***OR*** ambiente ***OR*** ferramenta) ***AND*** (programação ***OR*** algoritmo) ***AND*** (aprendiza* ***OR*** dificuldade ***OR*** Auxílio ***OR*** apoio ***OR*** suporte ***OR*** ajuda ***OR*** *feedback*)

Devido ao grande volume nas publicações internacionais a pesquisa foi reduzida aos artigos que continham alguns termos específicos em seus títulos ou resumos. A seguinte consulta foi aplicada aos artigos do IEEE:

> (“*Document Title*”: programming environments for novices ***OR*** "*Document Title*”: Supporting novice programmers ***OR*** "*Document Title*”: Tool Programming Novice Learners) ***OR*** (“*Abstract*”: programming environments for novices ***OR*** "*Abstract*”: Supporting novice programmers ***OR*** "*Abstract*”: Tool Programming Novice Learners)

As revisões sistemáticas realizadas por Keuning, Jeuring e Heeren (2018) e Junior e França (2017) também foram consideradas no levantamento dos trabalhos correlatos devido a atualidade desses trabalhos e a relevância deles em relação ao problema abordado nesta pesquisa.

Foram os seguinte critérios de exclusão: (*i*) artigos cujo foco principal não eram o ensino de programação; (*ii*) quando não se tratava de ambientes utilizados em aulas presenciais ou semipresenciais; (*iii*) quando não oferecia algum tipo de interação conversacional com o aluno; (*iv*) quando não possuía um ambiente de autoria do aluno; (v) quando se tratava de trabalhos relacionados ao ensino de robótica; (*vi*) quando se tratava de ambientes

relacionados a jogos educacionais; (*vii*) quando se tratava de ambiente com programação em blocos e que não abordavam o emprego codificação.

Assim, foram selecionados 45 artigos para uma análise mais detalhada dos quais 9 foram considerados diretamente correlatos e, portanto, serão apresentados neste capítulo.

4.1. Ambientes Analisados

Nesta seção serão apresentadas as principais características identificadas nos trabalhos correlatos.

4.1.1. Analogus

O *Analogus* (JÚNIOR; FECHINE; COSTA, 2009) é um ambiente de resolução de problemas de programação que visa aprimorar a habilidade do aluno iniciante em solucionar problemas por meio do raciocínio por analogia. Para isso, essa ferramenta apresenta duas importantes características:

- Recuperar as atividades previamente resolvidas pelo aluno e que sejam similares a atual;
- Ajudar o aluno iniciante a refletir sobre os aspectos que tornam suas atividades semelhantes.

No *Analogus* os aspectos que tornam uma atividade semelhante aos problemas já resolvidos são analisados e então é sugerido ao aluno um sistema de raciocínio baseado em casos que se recupera automaticamente um conjunto de atividades resolvidas pelo aluno e similares a atual. No instante em que essas atividades são recuperadas, um *chatterbot* inicia um diálogo com o aluno iniciante, discutindo os aspectos que levaram tais atividades a serem similares.

Diante dos problemas recuperados e do diálogo com o *chatterbot* (Figura 11), o aluno cria a solução do novo problema proposto então a submete para a avaliação do professor. O professor, por sua vez, efetua comentários sobre a solução. Assim que todas as considerações do professor forem atendidas, a solução é armazenada na base de casos do Analogus.

O cliente é uma interface Web, implementada através da tecnologia Google Web Toolkit (GWT). Já o servidor é formado por três módulos: o módulo RBC, o módulo chatterbot e o módulo interpretador de Python. O modulo RBC é responsável pela implementação do sistema de raciocínio baseado em casos para o domínio de problemas de programação. O módulo *chatterbot*, desenvolvido com uso da linguagem AIML e responsável pela implementação do sistema de diálogo que ajuda o aluno a refletir sobre os problemas similares.

Figura 12 - Visão Geral do Analogus.
Fonte: (JÚNIOR; FECHINE; COSTA, 2009)

A característica mais marcante do Analogus é o fornecimento de diálogo a partir de exercícios anteriores e então discutir com aluno sobre as similaridades entre os exercícios já resolvidos e o atual. Todavia, os outros tipos de dificuldades do aluno durante a realização de uma atividade não foram tratados. Também não foram apresentados neste ambiente algum recurso de acompanhamento ou suporte ao aluno no ambiente de codificação da solução.

4.1.2. AIIP - Ambiente Inteligente para Iniciantes em Programação

Gomes et al. (2011) apresentam AIIP (Ambiente Inteligente para Iniciantes em Programação), esse ambiente é composto de um módulo prático de ensino, um módulo teórico e um assistente capaz de agir de acordo com o comportamento do aluno fornecendo dicas e *feedbacks*.

O sistema AIIP é composto por 4 camadas: visão, sistema, negócio e dados. A camada de visão é responsável pela interação como o aluno através do assistente, um animador gráfico e o interpretador, além de disponibilizar acesso aos conteúdos e as atividades propostas.

O AIIP possui alguns objetos educacionais de aprendizagem implementados com intuito de ajudar o aluno em seu aprendizado, por exemplo, a exibição de dicas durante a resolução dos problemas, a apresentação de *feedback* a cada problema resolvido, além de um sistema de estatísticas apresenta ao usuário os dados sobre o número de exercícios resolvidos dentro da unidade e a média final calculada de acordo com parâmetros pré-estabelecidos nos exercícios.

Quando o aluno está trabalhando no modo teórico, o assistente não reage às solicitações, porém monitora seu progresso para que posteriormente, na hora de se trabalhar no modo pratico, tais informações sirvam como base para que o assistente realize inferências sobre o aprendizado do aluno. Uma vez que o aluno começa a trabalhar no modo pratico com objetivo de resolver problemas específicos armazenados no ambiente, o assistente começa realmente a atuar.

A ativação do assistente é realizada automaticamente de quatro formas complementares: (1) após cada ação de pressionar a tecla *Enter*, (2) após cada ação de salvar o código, (3) após cada compilação do programa e (4) após uma solicitação específica do aluno

Não foram detalhadas no trabalho as técnicas usadas para implementação no tratamento do diálogo nem os recursos computacionais utilizados na construção de cada camada.

4.1.3. BlueFix

Watson, Li e Godwin (2012) propõem o *BlueFix* como um ambiente destinado as dificuldades de alunos iniciantes no uso da sintaxe e semântica da

linguagem Java. No *BlueFix* os *feedbacks* são gerados a partir de mensagens de erros e de exemplos obtidos através de voluntários na web[1].

Para os autores a originalidade do *BlueFix* está em fornecer um conjunto de técnicas ao aluno num nível progressivamente crescente de *feedbacks*. Seu primeiro passo é encorajar um aluno a resolver um erro e depois ajustar dinamicamente o nível e o tipo de *feedback* fornecido caso o ele não consiga resolver.

Watson, Li e Godwin (2012) argumentam que a ajuda ao aluno pode ser dividida em três níveis crescentes de elaboração:

1. Fornecer ao programador uma breve mensagem do problema;
2. Fornecer breves explicações ou exemplos genéricos;
3. Fornecer um nível adicional de suporte com base em possíveis ações corretivas.

A Figura 12 ilustra o fluxo utilizado pelo BlueFix no fornecimento de

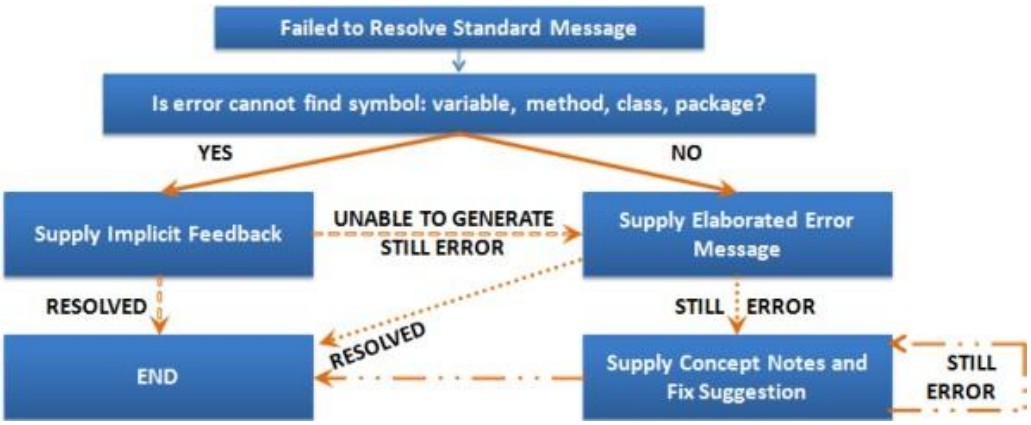


Figura 13 - Fluxo do feeback com base no erro do aluno no BlueFix.
Fonte: (WATSON; LI; GODWIN, 2012)

orientação ao aluno na correção do erro.

No ambiente BlueFix relevância do *feedback* fornecido é determinada a partir de 3 formas de ajuda:

[1] O termo original em inglês cunhado pelos autores é *Crowd-sourced.*

1. ***Feedback* implícito:** É considerado quando a causa mais provável do erro é comum. Nestes casos, sugerir uma correção que já foi aplicada no passado não é apropriada. Portanto, os autores propõem a construção de manipuladores específicos para cada tipo de erro conhecido onde uma verificação do código é realizada em busca de entender a causa do erro do aluno e assim ajudá-lo;
2. **Mensagens elaboradas de erro:** Ocorrem quando é possível determinar um feeback implícito adequado, então o erro é verificado numa base de 92 erros conhecidos e a ajuda é fornecida seguindo 9 princípios de clareza numa mensagem padrão do compilador, sendo eles: clareza, especificidade, insensibilidade ao contexto, localidade, fraseado apropriado, consistência, design visual adequado e ajuda extensível;
3. **Fornecimento notas conceituais e de sugestões:** Se as mensagens elaboradas de erros não ajudarem o aluno a resolver um erro, o *BlueFix* tentará localizar e apresentar uma correção de código, para isto, é buscado na base de dados um trecho de código que seja semelhante ao código-fonte do aluno.

O ambiente BlueFix limita-se ao fornecimento de dicas baseados na estratégia do aluno no código-fonte desenvolvido. Além do mais, depende que sejam criados distintos modelos, cada qual representado um caminho possível na resolução do aluno.

4.1.4. CodeMage

O ambiente *CodeMage* foi proposto por Whittall et al. (2017) para ser um ambiente de programação interativo voltado para alunos iniciantes em programação básica.

Whittall et al. (2017) afirmam que os ambientes de programação normalmente são projetados para programadores experientes e entendem que os alunos novatos gastam um tempo excessivo dominar e superar a complexidade de funcionamento dessas ferramentas avançadas, logo os alunos criam um medo espúrio em suas mentes sobre a programação em si.

Daí o *CodeMage* apresenta-se como alternativa ao implementar alguns recursos que os autores julgam facilitar a vida desses estudantes iniciantes. Os recursos são:

Editor de código: é onde o aluno poderá executar toda a codificação das tarefas. O editor tem funcionalidade de realce de sintaxe e realce de erro. A autoedentação e a sugestão de código também são suportados pelo editor.

Visualização da execução do programa (*debugger*): Permite que os alunos assistam à execução do programa e também fornece uma simulação visual do estado das variáveis ao executar linha por linha do programa;

Dicas de boas práticas: Neste recurso uma análise do código estático é realizada para determinar se houve erros no código ou presença códigos maliciosos, e então informa ao aluno dicas para o ajustamento da solução.

Seguem abaixo uma descrição dos recursos de interação com o aluno promovido pelo ambiente *CodeMage*:

- **Narrador:** é o recuso orientado à comunicação automática com o aluno via texto ou voz. A interface do CodeMage tem um pequeno espaço na parte inferior direita da tela onde o narrador reside, onde as mensagens são enviadas e apresentadas ao aluno;
- **Gerador de Código:** oferece blocos prontos de comando onde o aluno apenas clica sobre um bloco e a sua estrutura básica é levada para o código, cabendo ao aluno complementá-lo ou adaptá-lo de acordo com o contexto de sua atividade;
- **Tutor Remoto:** neste recurso o aluno poderá compartilhar em tempo real sua tela com uma outra pessoa, assim pode obter ajuda quando precisar;

De acordo com Whittall et al. (2017), o suporte no entendimento dos erros ocorridos é um dos principais mecanismos de auxílio ao aluno num ambiente de programação, pois permite que os iniciantes identifiquem e aprendam a superar seus erros e equívocos.

A riqueza de recursos na IDE é um ponto forte no *CodeMage*, destacando-se, sobretudo, as dicas de boas práticas de programação. Outro aspecto relevante é o fornecimento de informações amigáveis sobres os erros, pois propicia as condições de aprendizagem nestes momentos e propicia as condições para que o aluno supere as suas dificuldades e conclua as atividades propostas. Porém, não foi apresentado nenhum recurso para que o professor possa intervir na configuração do ambiente e o adeque ao seu estilo de mediação. Também não foram apresentados os recursos para que o aluno informe algum outro tipo de dificuldade ou que descreva a sua dúvida diretamente ao objeto ambiente.

4.1.5. LabPy - Laboratório virtual de ensino em Python

O *LabPy* (SIROTHEAU et al., 2018) é um ambiente interativo criado com o intuito de auxiliar os alunos no aprendizado de programação através da linguagem *Python*, sendo utilizado como uma ferramenta de auxílio ao professor para realizar as atividades em laboratório de programação.

Um diferencial do *LabPy* é permitir a avaliação automática e o *feedback* nas atividades práticas com o *Python* nas teóricas através da técnica de identificação de similaridade semântica implementado com o algoritmo n-gramas. Tal recurso, fornece um ganho significativo para o professor, liberando-o da atividade de correção dos exercícios propostos ao tempo que auxilia o aluno em sua aprendizagem.

Embora sua atualidade, o *LabPy* não oferece importantes recursos de *feedbacks* tais como: informações sobre os erros ocorridos, permitir que o aluno faça as suas perguntas em linguagem natural ou permitir que o professor atue online no acompanhamento das dificuldades ocorridas durante a resolução das atividades propostas. Além do mais, o recurso de *feeback* imediato apontado no *LabPy* somente atua sobre as respostas textuais e não foi informado o desempenho do uso do algoritmo *n-gramas* no tratamento dessas mensagens enviadas pelos estudantes.

4.1.6. Feeper - Feedback Personalizado

Segundo (ALVES; JAQUES, 2014) nome *Feeper* é um acrônimo entre as palavras "***Feed****back* ***per****sonalizado*", sendo a personalização o recurso tido como principal característica do ambiente. O *Feeper* foi desenvolvido para web e visa apoiar a aprendizagem de programação através da linguagem de programação Java, auxiliando o aluno nas atividades em sala de aula e extraclasse.

O *feedback* ao aluno ocorre de duas maneiras, de forma automatizada ou através da atuação direta do professor no ambiente. O *feedback* automático ocorre quando o aluno submete seu código para análise por um Juiz Online, então alguns testes previamente cadastrados pelo professor são realizados e se identificado a ocorrência de algum erro, uma mensagem de suporte é enviada para o aluno. A outra forma de ajuda é fornecida pelo professor, neste caso, o aluno seleciona um trecho de código que ele esteja com erro ou dificuldade e então submete juntamente com uma pergunta ao professor que então avaliará o contexto fornecido e a dúvida e por fim, enviará uma ajuda adequada. Durante o registro de uma dúvida o aluno pode realizar anotações vinculá-las às linhas do código correspondente ao trecho da dúvida. Outra possibilidade é o aluno marcar o um dado código como favorito para uma eventual futura consulta.

Embora o *Feeper* seja uma proposta aderente aos objetivos deste trabalho, o ambiente apresenta poucos recursos que permita a liberação do professor das demandas recorrentes de ajuda. Pouca ou nenhuma inteligência foi aplicada ao ambiente, de forma que o apoio automatizado fornecido ao aluno se limita aos casos de falhas nos testes cadastrados previamente pelo professor.

Uma característica que se destacou no ambiente *Feeper* foi a possibilidade de o professor personalizar as mensagens de ajuda de acordo o contexto da atividade. Outro recurso relevante é a possibilidade de o aluno enviar as suas mensagens ao professor vinculando-as a um trecho de código-fonte, o que permite destacar a linha na qual está ocorrendo a sua dúvida. Na perspectiva do suporte ao professor a característica mais relevante é a

facilidade do docente acompanhar a sua turma através de um painel de resultados por exercícios, possibilitando que o professor acompanhe num painel como seus alunos estão evoluindo na resolução das atividades propostas.

4.1.7. Portugol Studio + Plugin Tree Walkers

Para Raabe et al. (2015) o componente *Tree Walkers* é *plugin* integrado ao ambiente de desenvolvimento Portugol Studio e objetiva de ser um gerador de dicas durante a realização dos exercícios de programação por alunos iniciantes.

As dicas geradas no *Tree Walkers* ocorrem a partir da verificação do código do aluno com base nos erros já conhecidos e catalogados com os devidos *feedbacks*. Esse processo inicia com o estudante submetendo um código para o corretor. O compilador analisa o código e gera uma árvore sintática abstrata da solução do estudante. Em seguida ocorre a verificação da similaridade entre a solução do estudante e uma das soluções consideradas modelos para a resolução do problema.

Embora sua relevância ao fornecer as dicas baseadas nos *treewalkers* e numa solução modelo, o ambiente analisado não estende a sua capacidade de suporte ao atendimento de outras dúvidas dos alunos e não oportuniza ao professor a personalização das dicas fornecidas. Outro agravante é que o ambiente é unidirecional, ou seja, a análise do estado do aluno é feita em segundo plano e é enviado o suporte conforme a análise realizada em seu código, portanto o aluno não participa diretamente da enunciação de sua dificuldade.

4.1.8. Ask-Elle

O *Ask-Elle* (GERDES et al., 2017) é um ambiente que suporta o desenvolvimento gradual de programas funcionais simples em *Haskell*. O ambiente realiza o envio de *feedback* informando se os alunos estão ou não no caminho certo. Estes alunos podem pedir uma dica a qualquer momento ou sempre que estiverem confusos ou sem condições de avançar na resolução da

atividade, então uma dica sobre o próximo passo necessário no código é enviada em apoio ao aluno.

As ajudas e as dicas são geradas automaticamente a partir de um conjunto de soluções modelo do problema. A principal funcionalidade do *Ask-Elle* foi construída com base no rastreamento do código do aluno através do emprego de AST na identificação da estratégia empregada por este aluno com base em caminhos seguidos por outros alunos ao resolver a mesma atividade.

Toda solução modelo do *Ask-Elle* é enriquecida com anotações especiais semelhantes a um comentário de código-fonte, essas marcações serão utilizadas para mapear onde possivelmente o aluno estará num código e a dica correspondente a aquele momento.

O *Ask-Elle* pode ser acessado através de um navegador da web. Na página principal o aluno seleciona o exercício que deseja solucionar, ao desenvolver o programa o aluno pode verificar se ainda está a caminho de uma solução correta, pedir uma única dica ou todas as dicas possíveis sobre como proceder no dado estágio do código que ele se encontra.

O ambiente frontend lida com a interação com o aluno e é responsável por exibir as mensagens de *feedbacks*, enquanto que o backend cuida da lógica responsável pela geração do *feedback*. A Figura 13 ilustra uma visão

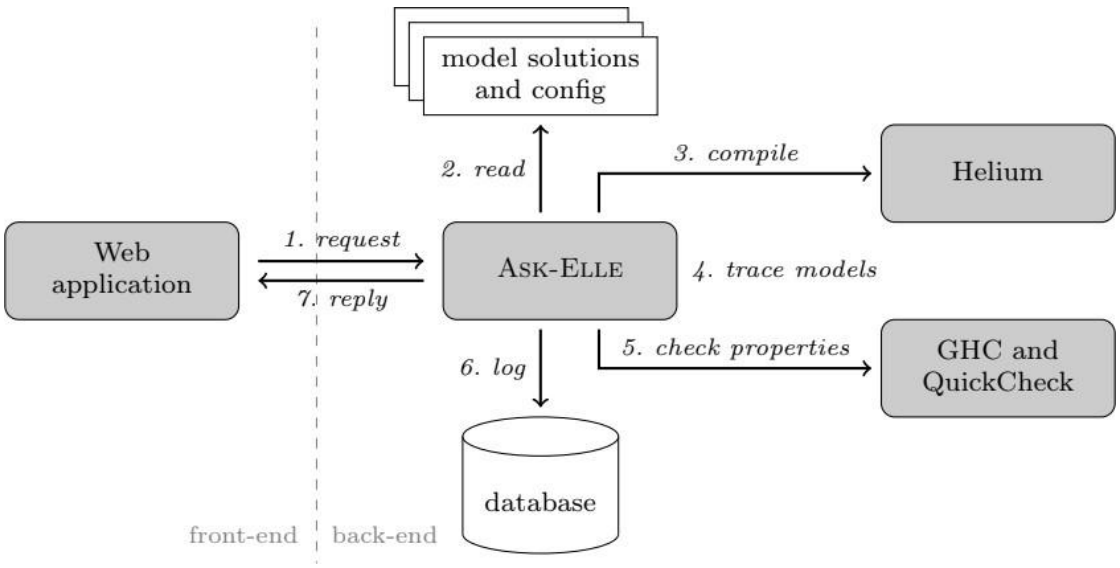


Figura 14 - Arquitetura do Ask-Elle.

Fonte: (GERDES et al., 2017)

geral da arquitetura do *Ask-Elle*.

No Ask-Elle a interface com o usuário é implementada como um aplicativo web usando a tecnologia *HTML* e *Ajax* para a requisição de serviços remotos. Cada vez que um aluno clica em um botão, como por exemplo o botão *Check* ou botão *Hint*, o *Ask-Elle* envia uma solicitação de serviço ao módulo raciocinador do domínio de programação (no *backend*), esta requisição por sua vez, contém todas as informações sobre o exercício que o aluno está resolvendo, a estratégia atual deste aluno e a sua última versão do código-fonte associado a dificuldade.

Os *feedbacks* são organizados numa hierarquia baseada na árvore sintática abstrata (*AST*) construída a partir das soluções modelos previamente registradas, isso permite que o professor configure o ambiente para o atendimento em níveis crescentes de detalhes ou de dificuldades. O *Ask-Elle* utiliza o compilador *Helium* para montar o programa e obter as mensagens de erros mais adequadas aos alunos iniciantes em programação.

Identificar a estratégia do aluno na resolução de um problema e fornecer as dicas apropriadas são as características mais marcantes no suporte ao aluno fornecido pelo *Ask-Elle*, porém, o ambiente preconiza a dica com base em modelos de soluções criadas pelo professor. Embora esse recurso permita que o professor diversifique os diferentes modelos e, por conseguinte forneça diferentes formas de ajuda, tal recurso cria um ambiente de difícil administração por parte do professor visto que o docente tem que idealizar as diferentes formas de resolver o mesmo problema, além do mais as dicas baseadas em modelos acaba por limitar a criatividade do aluno, pois indiretamente padroniza o seu comportamento aos moldes estabelecidos na "*forma de resolver*" do docente.

Embora o *Ask-Elle* utilize o compilador específico com o objetivo de fornecer as mensagens de erros de compilador de forma mais apropriadas aos alunos iniciantes, não foi possibilitado ao professor criar seus próprios *feedbacks* de erros e condizentes o nível do aluno e cada atividade proposta.

Outra limitação foi não permitir que o aluno externalizasse as suas dificuldades a partir da linguagem natural e que essas dificuldades fossem respondidas automaticamente pelo ambiente ou encaminhadas para a intervenção do professor.

4.1.9. Action Recommender

Nascimento et al. (2018) apresentam a ferramenta *Action Recommender* responsável por recuperar as mediações já utilizadas pelo professor e então recomendá-las aos alunos na ocorrência dos casos parecidos.

A ferramenta Action Recommender funciona como um plugin no ambiente *Moodle* e utiliza a técnica Raciocínio Baseado em Casos (RBC) empregada para fins educacionais. De acordo com Nascimento et al. (2018), o RBC permite que casos sejam empregados na representação das situações de dificuldades de aprendizado dos alunos e também as soluções adotadas por um professor no atendimento dessas dificuldades.

Para que ocorra a recomendação é necessário que o aluno preencha um questionário indicando um conjunto de atributos relevantes de sua demanda, então é realizado um cálculo de similaridade para cada caso registrado até que se encontre os casos semelhantes. Ao encontrar um caso semelhante as ações operadas neste caso análogo também são recomendadas ao aluno. Após recebimento de uma ajuda o aluno poderá dar um *feedback* se a ação pedagógica foi adequada e se por meio desse suporte ele conseguiu compreender o assunto. Se a ação pedagógica foi empregada com sucesso o caso solucionado será armazenado na base de casos e poderá ser reutilizado posteriormente, dessa forma o *feedback* do aluno permitirá *Action Recommender* realimente sua base de casos.

A ferramenta *Action Recommender* busca oferecer as condições para que o aluno solucione as suas dúvidas rapidamente e a partir de qualquer lugar, ao tempo que busca liberar o professor da tarefa de recomendação de objetos de aprendizagem condizentes com as dificuldades informadas.

Todavia, o ambiente não ofereceu um editor de código integrado, não avalia o código-fonte do aluno e além do mais as suas recomendações

somente acontecem se o aluno preencher adequadamente um formulário caracterizando sua dificuldade, do contrário o sistema pode não encontrar um caso semelhante de suporte para consultar as recomendações e enviá-las ao aprendiz.

4.2. Síntese dos Trabalhos Correlatos

A fim de possibilitar a análise dos trabalhos descritos na Seção 4.1 foi desenvolvido o Quadro 3 que reúne as principais características entre os trabalhos correlatos.

O fornecimento da ajuda relacionada aos erros cometidos pelos alunos foi destaque em 7 dos 9 trabalhos investigados. Normalmente a técnica empregada nesta análise foi a utilização das árvores sintáticas abstratas (AST). As outras técnicas empregadas no reconhecimento de erro residem na utilização de compiladores específicos ou no registro dos erros conhecidos, sendo este catalogo empregado diretamente na construção do ambiente ou através de recursos externos como uso compiladores específicos dedicados a alunos iniciantes.

Nos ambientes *Ask-Elle* e *TreeWalkers* o fornecimento de dicas ou de ajuda ao aluno frente a uma dificuldade foi realizada através do rastreio do código do aluno e através da comparação deste código com os modelos previamente criados pelo docente.

Somente os ambientes *Analogus* e *Action Recommender* forneceram um canal de diálogo com aluno em linguagem natural. No *Analogus* esse recurso foi implementado através do *AIML* e durante o diálogo era realizada uma conversação com o aluno sobre as semelhanças entre a nova atividade proposta e outras atividades já desenvolvidas por este aluno. No *Action Recommender* o aluno apenas registrava a sua dúvida num formulário e então o enviava juntamente outras informações necessárias para a caracterização do contexto da dúvida.

Embora a afetividade não seja um objeto de estudo desse trabalho, foi identificado o emprego desse recurso nos ambientes *LabPy* e no *Analogus*. O

emprego de um *feedback* com afetividade tem apontado efetivo no ganho da empatia e atenção do aluno na assimilação do suporte fornecido, e tem permitido uma interface homem-máquina mais harmoniosa.

Foi observado o emprego de casos de testes automatizados e o fornecimento de *feedback* durante a ocorrência de falhas nestes testes, os trabalhos realizados no *Feeper*, *Tree Walkers* e no *CodeMage* implementaram este recurso.

Os *feedbacks* apresentados nos trabalhos correlatos possuíam um contexto pré-formatado, normalmente o código ou os erros ocorridos, sendo ignorado, portanto a própria conversação ou a enunciação do aluno ao expressar a sua dúvida. O histórico de atendimento do professor não foi explorado no atendimento de casos recorrentes, exceto no *Action Recommender*, entretanto, neste ambiente o código do aluno e os erros ocorridos foram ignorados ao se analisar a dificuldade do aprendiz, sendo necessário que ele preenchesse um longo questionário informando os critérios necessários à classificação de seu caso de dificuldade, em detrimento a uma análise textual da dúvida expressa pelo aluno num possível ambiente de conversação.

Embora os distintos tipos de *feedback* fornecidos pelos trabalhos correlatos, não foi encontrado neles nenhum mecanismo planejado para permitir a extensão de suas funcionalidades. Uma exceção, no entanto, foi o ambiente Portugol que permitiu o acoplamento do plugin *Tree Walkers* estendendo assim sua capacidade, embora o objeto de estudo tenha sido o *plugin* chamado*Tree Walkers*.

Nos ambientes pesquisados normalmente o aluno não expressava as suas dúvidas diretamente numa conversação. Prevaleceu nos ambientes uma comunicação unidirecional onde o aluno era apenas um receptor. A única exceção foi o ambiente *Analogus* que permitiu o diálogo sobre as semelhanças entre as atividades já desenvolvidas pelo aluno e a atual, e também tratava possíveis dúvidas dos alunos. No entanto, o *Analogus* não apresentou os recursos necessários para se estender a capacidade de atendimento do ambiente e também não demonstrou os mecanismos que permitisse a

expansão do leque de cenários de atendimentos para outros tipos de dificuldades.

4.3. Relações entre os Trabalhos Correlatos e a Proposta desta Dissertação

A proposta deste trabalho se diferencia por oferece um ambiente de programação com dois tipos principais de *feedback*, aqueles orientados as dificuldades informadas pelo aluno através de uma área de conversação em linguagem natural via texto, onde o aluno terá a liberdade de expressá-las ao seu modo, diferentemente dos trabalhos correlatos onde essas dificuldades são apenas identificadas através da análise em segundo plano ou pelo preenchimento de formulário. O outro tipo de feeback oferecido neste trabalho refere-se aos relacionados a código-fonte do aluno, embora outros trabalhos realizem diversos tipos de suporte neste quesito, este trabalho se diferencia por oferecer flexibilidade para o professor personalizar os *feedbacks* oferecidos.

A fornecimento de suporte personalizado por dificuldades esperadas numa atividade normalmente foram tratados através da criação de soluções modelos, RBC, AST ou através da análise de exercícios análogos, nossa proposta prever que o professor possa utilizar assistentes específicos com a capacidade de processamento de linguagem natural e de análise de código-fonte, cada qual especialista numa dificuldade e com mecanismos próprio de inferência. Ao modelar as dificuldades esperadas em termos desses assistentes o professor oferecerá um suporte dinâmico, podendo a qualquer momento remover ou adicionar novos assistentes, sendo que as evoluções desses agentes serão autônomas, permitindo um ambiente dinâmico e com aprendizado.

Nossa proposta prever extensibilidade ao ambiente ao permitir que novos assistentes sejam acoplados estendendo a área de suporte prevista inicialmente. Outro diferencial refere-se à capacidade do ambiente aprender com o suporte oferecido pelo professor, podendo ao longo do tempo possuir os mesmos estilos de mediação do docente à medida que evolui a sua capacidade de atendimento.

Quadro 3 - Comparação entre trabalhos correlatos.

	Action Remember	Analogus	Ask-Elle	AIIP	BlueFix	CodeMage	Feeper	LabPy	Tree Walkers	Proposta
Linguagem	-	Python	Haskell	Baseada em Pascal	Java	Java	Java	Python	Portugol	Python
Técnica para reconhecimento da dificuldade	RBC	RBC e AIML	AST	Métricas	Análise de Erros	Análise de Erros	Testes automatizados	N-gramas	AST	Agentes inteligentes com a capacidade de análise de código-fonte e processamento de linguagem natural.
Prever expansão	Não	Não	Não	Não	Não	Não	Não	Não	Sim	Sim
Possui editor de código integrado	Não	Sim	Sim	Sim	Não	Sim	Sim	Sim	Sim	Sim
Ambiente online	Sim	Sim	Sim	Não	Não	Sim	Sim	Sim	Não	Sim
Flexibiliza configurar ajuda por exercícios	Não	Não	Sim	Não	Não	Não	Sim	Não	Sim	Sim
Contexto utilizado no apoio	Formulários	Atividade	Código-fonte	Código-fonte; Erros; Histórico de Navegação	Código-fonte	Erros gerados	Código-fonte	Código-fonte; atividade	Código-fonte	Código-fonte; Atividade;Discurso do aluno.
Aprende com os de atendimentos anteriores	Sim	Não	Não	Não	Não	Não	Não	Sim	Não	Sim
Responde as dúvidas registradas	Sim	Sim	Não	Não	Não	Não	Não	Sim	Não	Sim
Explica as causas de erros	Não	Não	Sim	Sim	Sim	Sim	Sim	Sim	Sim	Sim
Guia à resolução a cada passo	Não	Não	Sim	Sim	Sim	Não	Não	Não	Não	Não
Aluno interage com o professor ou colegas	Sim	Não	Não	Não	Não	Não	Sim	Sim	Não	Sim, com o professor
Exibe avaliação da atividade	Não	Não	Não	Sim	Não	Não	Não	Sim	Não	Sim, com notas por habilidades desenvolvidas.
Trata afetividade	Não	Sim	Não	Não	Não	Não	Não	Sim	Não	Não
O professor recebe e atua dificuldades não resolvidas pelo ambiente	Sim	Não	Não	Não	Não	Não	Não	Sim	Não	Sim e utiliza essa atuação para aperfeiçoar o ambiente.
Possui avaliação automática da solução	Não	Não	Não	Não	Não	Não	Não	Sim	Não	Não
Possui testes automatizados	Não	Não	Não	Não	Não	Sim	Sim	Não	Sim	Sim

Fonte: próprio autor.

5. Ambiente Proposto

Neste capítulo serão descritos os principais aspectos que envolvem a construção do ambiente de programação proposto, denominado aqui por *AmPaRe* (***Am***biente de ***P***rogramação com Suporte aos ***A***tendimentos de Demandas ***Re***correntes).

Inicialmente, na Seção 5.1 será apresentada a visão geral do escopo do problema.

Em seguida, na Seção 5.2 será uma visão geral da solução proposta, onde serão descriminados os atores do sistema e as principais funcionalidades.

Logo após, na Seção 5.3 serão abordados a proposta de suporte a mediação pedagógica baseada em assistentes inteligentes.

Posteriormente, na Seção 5.4 serão descritas as funcionalidades de suporte a mediação do professor e de preparação das atividades de programação.

Já na Seção 5.5 serão abordados os tipos de *feedbacks* que serão oferecidos pelos assistentes no auxílio ao aluno na realização das atividades de programação, os recursos previstos para o ambiente de codificação e o suporte à autorregulação do aluno ao desenvolver suas atividades.

Na Seção 5.6 será apresentada a modelagem estrutural da arquitetura proposta, sendo abordada a organização dos recursos do sistema sob a perspectiva da engenharia de software e do emprego dos padrões de projetos, ambos aplicados com o objetivo de propiciar a extensibilidade da solução na anexação de outros assistentes desenvolvidos externamente.

Por último, na Seção 5.7 serão apresentadas as considerações finais deste capítulo.

5.1. Visão Geral do Escopo do Problema Tratado

A proposição de exercícios para os alunos iniciantes em programação exige um grande esforço do professor no provimento do suporte requerido por estes aprendizes. Por outro lado, o aluno é envolvido por variados tipos de dificuldades que os impedem de completarem essas atividades propostas, além do mais, o

repositório de atividades, de conteúdo e o ambiente de programação são distintos, exigindo do aluno o acesso a variados ambiente para o desenvolvimento de uma única atividade.

As dificuldades dos alunos (Figura 14) tratadas nesta proposta de trabalho são aquelas decorridas a partir da tentativa de resolução das situações-problemas propositadas pelo docente e que o aluno estará resolvendo nas aulas de laboratório ou em continuidade fora do espaço acadêmico.

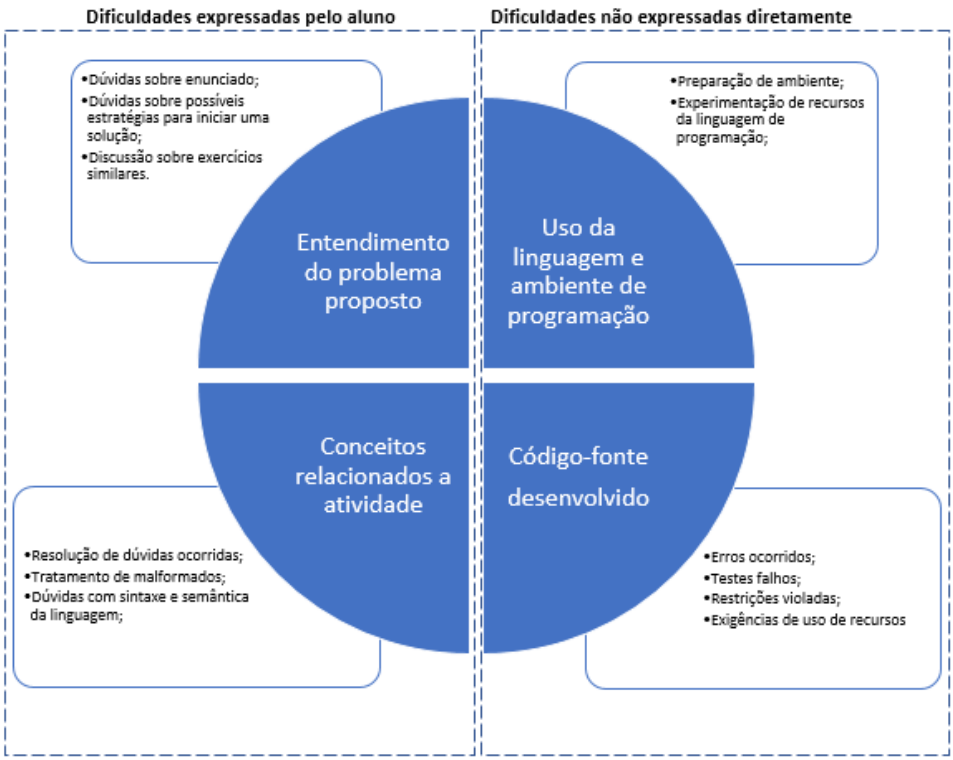


Figura 15 - Visão geral das dificuldades dos alunos iniciantes durante as atividades de programação.

Entendemos que essas dificuldades, são aquelas expressadas diretamente pelo aluno ao enunciá-las, por exemplo as dificuldades relacionadas ao entendimento do problema proposto, sobre os conceitos desenvolvidos nas aulas teóricas e que poderão ser utilizados na atividade, dúvidas sobre decorridas dos conflitos ocasionados pela proposição do problema dentre outras. Já as dificuldades não são expressadas diretamente pelo estudante, normalmente aquelas associadas aos erros ocorridos na tentativa de resolução da atividade, violações de restrições

previstas no código, as verificadas pelo não cumprimento de exigências uso recursos da linguagem na solução, bem como as dificuldades com a preparação do ambiente de programação.

Entendemos que parte das dificuldades dos alunos são repetitivas para o professor, todavia, o docente não é liberado dessas demandas recorrentes e suas condições de atendimento são limitadas por razões de tempo em sala de aula e também pelos aspectos físicos-geográficos quando essas dificuldades ocorrem fora do laboratório, no entanto, esse suporte é de máxima valia para o aluno iniciante quando se encontra com dificuldades que, mesmo sejam simples ou corriqueiras, os impedem de avançarem na resolução do problema proposto.

5.2. Visão Geral da Solução Proposta

A proposta de dessa trabalho é a criação de um ambiente de desenvolvimento onde o aluno poderá encontrar num só lugar os recursos necessários para: (*i*) acessar o repositório das atividades; (*ii*) desenvolver e executar os seus programas; (*iii*) obter suporte personalizado para as dificuldades manifestadas diretamente pelo aluno ou aquelas identificadas através da análise do código-fonte desenvolvido; (*iv*) acompanhar as suas atividades avaliadas; e principalmente, (*v*) ser atendido a qualquer tempo por assistentes inteligentes disponíveis para as dificuldades previstas ou ser atendido pelo professor nos demais casos.

Outros recursos foram propostos no ambiente do aluno com o objetivo de facilitar o desenvolvimento e a experiência com a linguagem de programação, dentre eles incluem: (a) console para testes isolados, sem comprometer a atividade em desenvolvimento; (b) auto complemento da sintaxe; (d) destaque nas palavras chaves da linguagem e versionamento de código-fonte.

Para o professor são previstos os recursos para: (a) catalogar as atividades propostas; (b) acompanhar o desenvolvimento das atividades; (c) avaliar as atividades através das habilidades desenvolvidas; e principalmente, (d) transferir parte de seus atendimentos de suporte para o ambiente a fim de ser liberado das demandas recorrentes.

A construção da atividade (Figura 16) é, em primeira instância, onde o professor irá propor uma situação problema e então registrar o suporte previsto para a atividade. Neste momento o docente poderá agregar a sua bagagem histórica

sobre as dificuldades que possivelmente os alunos terão ao realizarem a atividade proposta, ao tempo que o professor delegará ao ambiente aqueles atendimentos recorrentes, podendo então se dedicar as outras atividades de mediação e tratar somente as situações mais graves ou não solucionadas pelo ambiente.

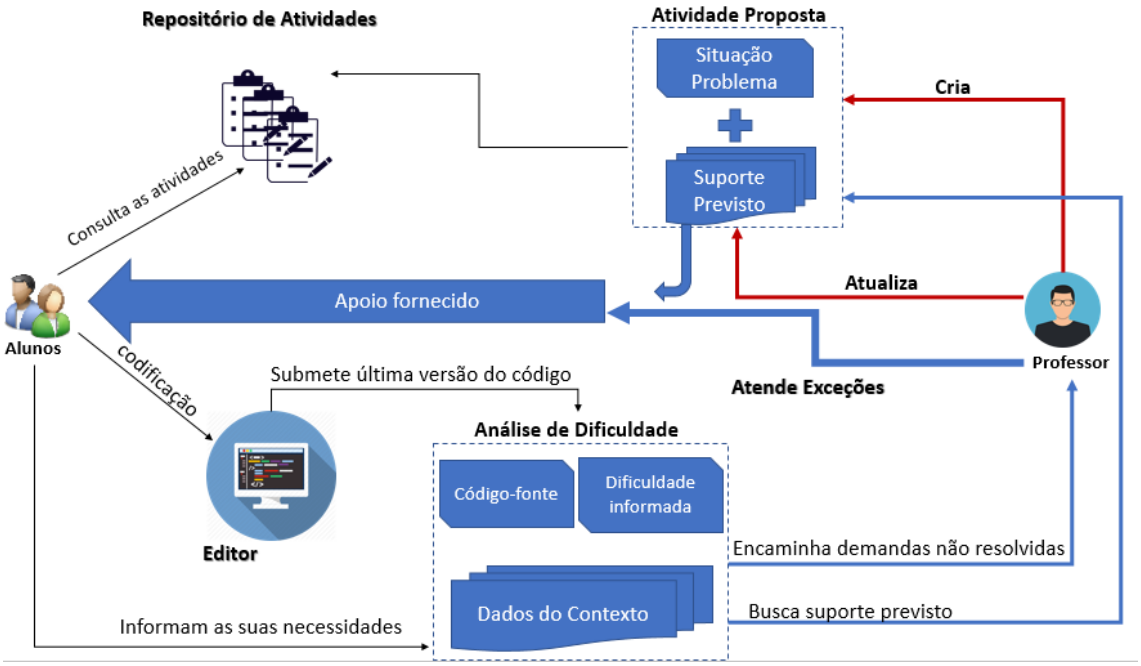


Figura 16 - Visão geral do ambiente proposto.

Fonte: Autoria Própria

Ao aluno é previsto o acesso centralizado as atividades propostas, as avaliações recebidas e principalmente ao ambiente de codificação e de conversação sobre as dificuldades que possivelmente ocorrerão.

Como ilustra a Figura 16, ao iniciar uma atividade o aluno será encaminhado ao ambiente de codificação (editor), a partir desse momento o aluno poderá dialogar sobre as suas dificuldades, seja através da enunciação direta do pedido de suporte ou pela conversação que poderá ser iniciada pelo próprio ambiente ao analisar o código do aluno e identificar algum indício de dificuldade.

5.3. O Suporte a Mediação Pedagógica Baseadas em Assistentes Inteligentes

> O ambiente proposto visa propiciar ao professor os recursos para que o docente delegue uma parte de seu atendimento a quem se denominou aqui de assistentes, sendo os assistentes conversacionais (ACs) e os assistentes analisadores de códigos (ACOs).

Os assistentes deverão ser constituídos a partir do emprego de agentes inteligentes que lidarão com o conhecimento prévio do professor e com as demandas de suporte decorridas durante a realização das atividades propostas. Neste contexto, buscou-se manter sempre preservado a autonomia professor ao delegar os atendimentos a esses assistentes ao mesmo tempo que será fornecido a autonomia necessária para que esses assistentes, enquanto agentes inteligente, interajam e atuem segundo os seus mecanismos internos de inferência na sua tomada de decisão ao lidar com o ambiente.

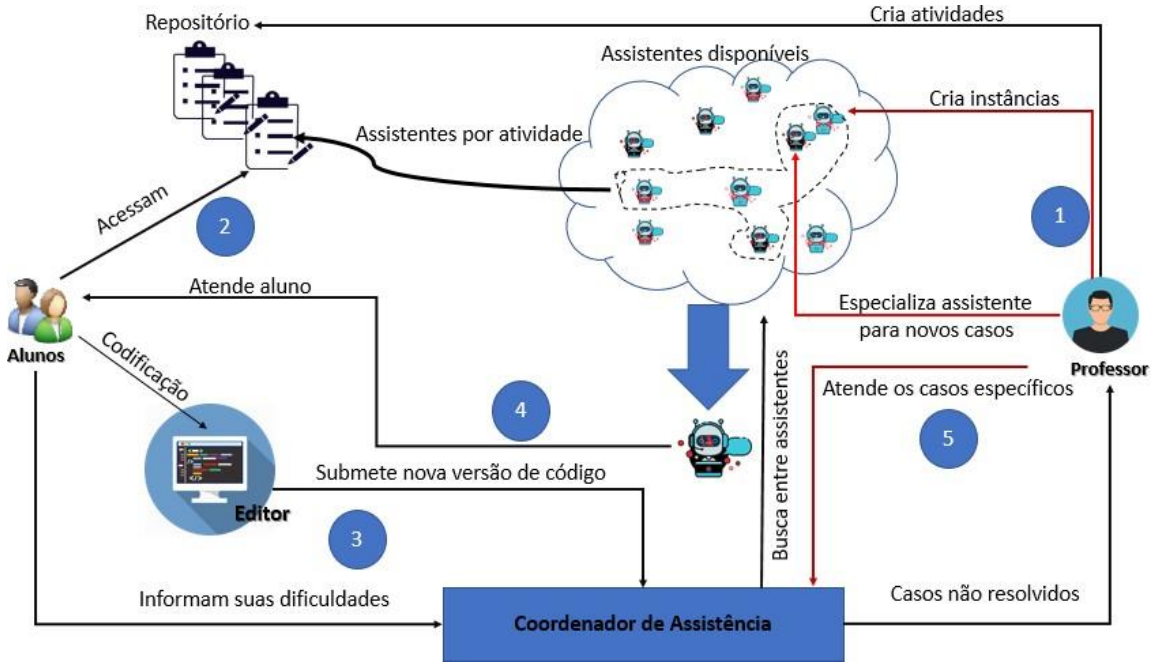


Figura 17 - Ciclo de suporte previsto durante uma atividade.

Fonte: Autoria Própria

A modelagem do suporte previsto para uma atividade é realizada a partir da seleção dos assistentes entre os disponíveis no ambiente e que serão encarregados pelo acompanhamento da atividade (passo 1 da Figura 16).

Ao iniciar uma atividade (passo 2 da Figura 16) o aluno deverá ser encaminhado para área de codificação e a partir do desenvolvimento da atividade o aluno poderá iniciar um diálogo solicitando um *feedback* ou ao salvar ou executar uma nova versão se seu código (passo 3 da Figura 16) um diálogo com o aluno poderá ser iniciado.

O módulo *Coordenador de Assistência* será responsável por propagar a demanda de suporte ou o código fonte para a análise pelos assistentes disponíveis, onde cada um desses assistentes analisará isoladamente a demanda em busca de uma ação de suporte. Na ocorrência de assistentes para a dificuldade ocorrida, é papel do coordenador de assistência interfacear essa comunicação com o aluno, na Seção 5.6 será abordado em detalhes as estratégias arquiteturais para essa comunicação.

Ao propagar para os assistentes a demanda de suporte cada um deles realizará a análise das possíveis dificuldades e de sua capacidade em oferecer um suporte mais adequado, essa condição do assistente deverá ser informada em termos de probabilidade variando entre 0 e 100%, em seguida esse índice será utilizado pelo Coordenador de Assistência ao organizar uma fila de atendimento e selecionar o assistente mais adequado para a conversação (passo 4 da Figura 16).

Caso não haja ao menos um assistente para o atendimento ou caso as assistências fornecidas tenham sido mal avaliadas pelo aluno, a demanda será transferida para o professor que decidirá a forma como prestará o seu atendimento, se pelo sistema ou presencialmente (passo 5 da Figura 16).

5.3.1. Os Tipos de Assistentes Inteligentes

Os assistentes conversacionais (ACs) serão constituídos por agentes inteligentes especializados no tratamento das dificuldades informadas pelo aluno através do ambiente de conversação via texto. Dentre as dificuldades que serão atendidas pelos ACs estarão as dúvidas sobre o enunciado do exercício proposto, as dúvidas sobre os conceitos já ministrados em aulas anteriores, as dificuldades associadas ao

correto uso de recursos da linguagem, tais como sintaxe e a semântica de comandos. Outros casos de atendimento poderão ser especializados pelo professor conforme demanda ou experiência do docente na disciplina.

Os assistentes analisadores de código (ACOs) também serão constituídos a partir de agentes inteligentes, todavia, nos ACOs eles analisarão o código-fonte da resolução do aluno em busca de eventos que determinem a necessidade de suporte. O código-fonte será obtido durante a submissão da atividade, sempre ao salvá-lo ou durante a tentativa de execução do programa desenvolvido. Os ACOs possibilitarão, dentre outros *feedbacks*, informar ao estudante as mensagens de erros conforme mediações previstas pelo professor, de forma que os erros normalmente técnicos e não orientados ao ensino, sejam mais amigavelmente comunicados, orientando a reflexão e o aprendizado do estudante.

Os ACOs também poderão verificar o código do aluno em busca de restrições e violações de regras da atividade, por exemplo, a obrigatoriedade de uso de certo operador ou da violação de restrição de uso de certo recurso negado pelo professor para aquele momento da turma, outro exemplo ocorre quando o docente requer a construção do algoritmo com tal funcionalidade e o aprendiz decidiu por usar um recurso externo já pronto sem codificar a solução solicitada. Os ACOs poderão ser dedicados aos testes da solução com valores de entrada e de saída pré-definidos pelo professor e então fornecer *feedbacks* específicos para os casos onde esses testes falham.

5.4. O Suporte a Mediação e à Autoria do Professor

Em se tratando de introdução a linguagem de programação as necessidades de mediação são evidentes em cada atividade desenvolvida e a ausência desse suporte reverbera imediatamente no progresso e no desempenho dos alunos.

É difícil afirmar que alguém ensina alguém a fazer algoritmo e é nocivo ao processo um ensino bancário, onde o professor apenas se apresenta como alguém que *“dá aula”* e que “*dá notas nas atividades*”. Não se pode pensar numa situação na qual o professor ensina algoritmos fazendo com que os aprendizes adquiram o estilo do professor de programar, o contrário disso exige tempo de qualidade para que o professor se dedique ao acompanhamento do aluno, ao entendimento da

forma de concepção e de resolução de um dado problema pelo aprendiz, de tal forma que educador se torne educando e o educando um educador de suas práticas e de sua forma de ver o mundo através do problema proposto.

A proposição de problemas tem sido uma importante ferramenta aplicada nas aulas em laboratório, mas a ação mediadora do professor não é plenamente realizada durante essas atividades, hora, pela qualidade dos exercícios propostos, que muitas vezes são orientados para toda a turma, ao invés de ser direcionados a um indivíduo como um fator desequilibrador de suas certezas e causador de dúvidas. Outrora, pela minimização dos conflitos, afinal o professor não tem muito tempo, nem para "dá o básico", então os exercícios que provocam os desafios, os erros e os desiquilíbrios do aluno são muitas das vezes evitados e os cenários que causam as eventuais dúvidas são reduzidos, logo as condições de aprendizagem são também atenuadas.

De acordo com Yacef (2002), auxiliar os professores e os instrutores a lecionar melhor é uma atividade tão importante quanto ensinar os alunos. Logo, foi entendido que as ferramentas adequadas ao ensino tanto devem permitir ao professor criar as estratégias de mediação quanto liberá-lo das demandas repetitivas, de forma que ele, o professor, possa concentrar a sua atenção nos casos mais críticos ou de maior relevância para o grupo ou indivíduo.

Os recursos do *AmPaRe* visam facilitar a mediação do professor através dos seguintes recursos:

- O professor poderá cadastrar as habilidades que serão desenvolvidas no curso e distribuí-las durante as atividades propostas: busca-se com este recurso que docente possa acrescentar um gradativo grau de dificuldade à medida que novas atividades são propostas e que além disso ele possa construir atividades que explorem os conhecimentos já adquiridos pelo aluno e que o aprendiz venha ganhar mais autonomia no passo que novos conteúdos sejam inseridos e os desafios sejam propostos;
- O professor poderá identificar as questões com maiores demandas de ajuda ou que possuem a maior incidência de erros, então o docente

poderá antevê as ações de mediação, tanto através do suporte pelos assistentes quanto pelas proposições de novos conteúdos e exercícios de revisão para que essas dificuldades sejam minimizadas;

- A avaliação da situação do aluno é contínua e por atividade, pois cada dificuldade apontada pelo estudante é registrada e poderá ser consultada a qualquer tempo;
- As avaliações realizadas pelo professor deverão ser acompanhadas de mensagem para o aluno, possibilitando que o professor estimule o desenvolvimento do aprendiz;
- Deverá ser dada a autonomia para professor intervi em qualquer atendimento fornecido pelos ACs e ACOs, podendo o docente complementar o suporte fornecido por um destes assistentes a qualquer momento;
- O professor deverá ser notificado de casos onde os estudantes não foram bem atendidos por um assistente, assim o docente poderá deslocar seletivamente a sua atenção aos casos de exceção e as situações mais graves ou mesmo responder a esses eventos diretamente através do ambiente;
- Os ACOs deverão apoiar o professor na explicação de causas de erros liberando-o de explicações repetitivas ao tempo que oferecerá diálogos que propicie a reflexão e o aprendizado do aluno a cada erro ocorrido;
- Os ACOs deverão apoiar o docente na identificação de violações de restrições estabelecidas, apontando diálogos específicos por restrições e contextualizadas à atividade;
- O professor poderá registrar mensagens específicas para os casos de falhas de testes nos programas dos alunos, essas mensagens poderão ser diferenciadas por cenários de testes ou criadas pelos mecanismos de atuação dos assistentes.

5.4.1. A Modelagem de Suporte Previsto para a Atividade

A preparação do suporte previsto será realizada a partir da seleção de assistentes disponíveis no ambiente e que ficarão encarregados pelo tratamento das dificuldades ocorridas na atividade, logo, uma atividade deverá ser descrita em

termos dos assistentes encarregados pelo atendimento das dificuldades ocorridas durante as tentativas de resolução.

Com o conhecimento prévio sobre o comportamento cada assistente o professor, ao criar ou dá manutenção numa atividade, será responsável por escolher estrategicamente os assistentes que guiarão a mediação pedagógica esperada. A Figura 17 ilustra um processo de seleção instâncias específicas de assistentes que monitorarão as demandas de suporte.

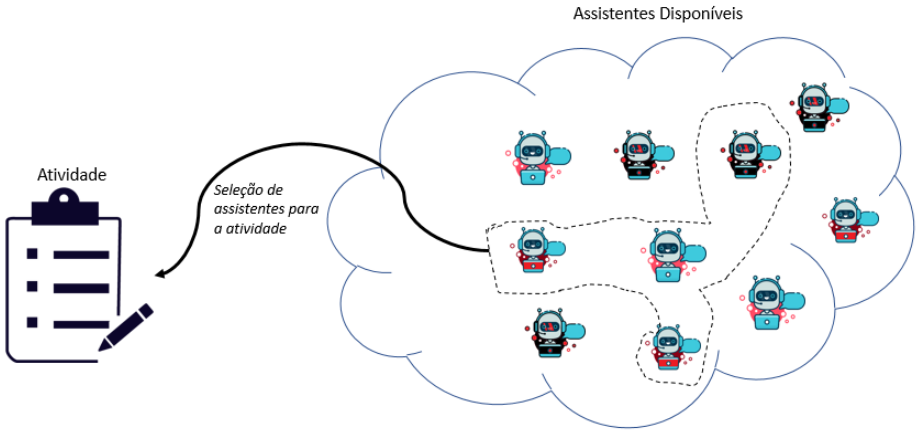


Figura 18 - Modelagem de suporte previsto na atividade.

Fonte: Autoria Própria

Um variado número de assistentes poderá trabalhar competindo por um atendimento e ao mesmo tempo cooperando no fornecimento de condições para que o estudante se desenvolva durante a realização da atividade proposta. Assim que novos assistentes foram incorporados ao sistema ou ganharem novas capacidades de suporte o professor poderá voltar a atividade e redefinir os assistentes responsável por acompanha-las.

5.5. O Suporte às Dificuldades do Aluno e sua Autorregulação

No Capítulo 2 foi realizada a revisão a respeito da importância do diálogo e do acompanhamento do aluno durante as atividades de programação. Destacou-se nestes estudos o fato que o aluno iniciante de programação é carente do suporte conversacional durante a realização das atividades propostas. Talvez um cenário ideal fosse aquele onde cada estudante pudesse ser acompanhado por um

professor, mas por limitações de recursos humanos e de custo esta não é uma solução viável.

Um *feedback* é conceituado como as informações fornecidas por um agente, por exemplo, o professor, um colega, o livro, pai, experiência e ou a própria reflexão sobre um dado assunto. Segundo HATTIE e TIMPERLEY (2007), o *feedback* ocorre em relação aos aspectos do desempenho ou da compreensão de uma pessoa. No contexto desse trabalho, um feeback é entendido como o apoio fornecido, seja uma resposta motivada por uma pergunta do aluno ou em reação dificuldades identificadas ao analisar o código do aluno.

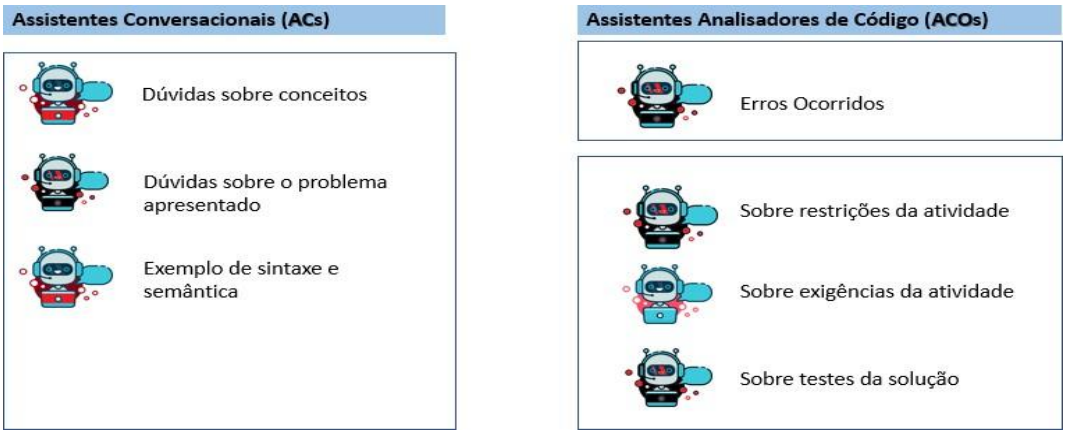


Figura 19 - Família de assistentes previsto por tipo de suporte.
Fonte: Autoria Própria

Os tipos de *feedback* previsto foram ilustrados na Figura 18 como uma família de assistentes instanciados a partir do comportamento base dos assistentes conversacionais (ACs) e dos assistentes analisadores de código (ACOs).

Nos baseando nos tipos de *feedbacks* elencados por Keuning, Jeuring e Heeren (2018), os assistentes conversacionais (ACs) disponibilizarão atendimentos para as seguintes necessidades:

- **Dúvidas sobre os conceitos estudados:** Essa necessidade ocorre com frequência durante uma atividade quando o aluno se encontra em situações de desequilíbrios cognitivos. De acordo com Nevado, Menezes e Júnior (2011) é fundamental a criação de atividades que

provoquem os desequilíbrios cognitivos, todavia, é igualmente importante que ao mesmo tempo seja oferecido o suporte para que ocorra as reconstruções, atitudes ativas e reflexivas. É comum, portanto, que ao ser conflitado o aluno busque revisar os seus conceitos e assimilá-los dentro do seu novo saber. Identifica-se que durante a realização das atividades o aprendiz acaba por evidenciar os seus conceitos malformados e quando ciente destes conflitos é observado que estes aprendizes necessitam revisar adequadamente seus pontos de dúvidas. Ocorre que a revisão literária nem sempre é acessível rapidamente, sendo necessário que o aluno se volte a extensas apostilas ou livros, desmotivando assim a busca pelo conhecimento. Ao atender as dúvidas registradas pelo estudante o *AmPaRe* permitirá que essa consulta seja feita rapidamente através no ambiente de conversação;

- **Dúvidas sobre o problema apresentado:** Frequentemente o professor interrompe uma ajuda para novamente comunicar a turma uma explicação do enunciado ou maiores detalhes sobre o que se espera do comportamento do programa. Isto ocorre devido as distintas perspectivas dos estudantes frente à tarefa e seus diferentes níveis de análise sobre problema proposto. A criação de um enunciado com muitos detalhes pode ser boa para alguns, mas ao mesmo tempo pode prejudicar outros alunos por acharem a tarefa volumosa e complicada. O ambiente proposto deve permitir que o professor especifique os diferentes níveis de detalhes e exemplos sobre o comportamento programa, de forma que o aluno possa realizar essas consultas por demanda através do ambiente de conversação com os ACs;
- **Necessidade de exemplo de sintaxe e semântica:** As dúvidas sobre a sintaxe e semântica no uso dos recursos da linguagem é comum durante as primeiras aulas de programação, momento no qual o aluno está aprendendo uma nova linguagem e precisa se familiarizar-se com ela e com seus recursos. Essas dificuldades retornam sempre que é introduzido um novo conceito ou quando são promovidas as atividades para o desenvolvimento dessas. Essas dúvidas são demasiadas e

ocorrem ao mesmo tempo e por uma gama de alunos, impossibilitando que o professor dedique atenção a cada caso. É comum ouvir dos aprendizes "*eu até sei o que preciso usar, mas não sei como*". É proposto no ambiente um tipo de AC dedicado a este tipo de dificuldade. Diferentes instâncias do AC dedicados a oferecer exemplos de sintaxe e semântica podem ser especializadas, cada qual dedicado a uma dada dificuldade ou mesmo especializando os exemplos conforme o contexto da atividade em desenvolvimento;

Baseado nos tipos de *feedbacks* estabelecidos por Keuning, Jeuring e Heeren (2018), os assistentes de analisadores de código (ACOs) disponibilizarão suporte para as seguintes atividades:

- **Explicação sobre as causas de erros ocorridos:** A realização das atividades de programação é frequentemente acompanhada das ocorrências de erros. Por conta do grande volume de erros o professor fica impossibilitado de oferecer apoio e por consequência perde um importante momento de mediação pedagógica. Adicionalmente, é verificado que os erros fornecidos por interpretadores e compiladores não são orientados a aprendizagem, inviabilizando que o estudante aprenda e progrida com eles. Muitos desses erros são vistos pelos alunos apenas como um obstáculo, sendo comum as seguintes manifestações: "*o que faço para resolver isto?*", "*o programa está certo, só tem um erro me atrapalhando e não consigo encontrá-lo*" ou dizeres como "*Já tentei de tudo e não acho o erro!* ". Os assistentes responsáveis pela explicação de causas de erros do *AmPaRe* serão acionados a análise do código ou durante a execução do programa desenvolvido. As mensagens de erros são complementares aos erros fornecidos pelo interpretador e compilador, de forma que na disponibilidade de ambos o aluno possa também se familiarizar com as mensagens técnicas fornecidas pelos interpretadores ou compiladores da linguagem. Ficará a critério do professor personalizar ou não as mensagens de erros fornecidas de acordo com as instâncias de cada ACOs disponíveis na atividade;

- ***Feedbacks* sobre restrições na atividade:** Um assistente poderá estar dedicado ao acompanhamento de restrições violadas, tais como o uso de certos recursos numa atividade. Eles poderão identificar certos padrões no código-fonte e então iniciar uma conversação com o aluno a fim de lembrá-lo das restrições estabelecidas. Observa-se, que sem as ferramentas adequadas o professor somente terá o conhecimento da violação de restrições no código apenas quando ele estiver avaliando a solução final da atividade e esse *feedback* tardio acaba não causando o efeito na revisão das práticas desses alunos no momento de suas falhas. Há casos onde os alunos fogem de uma implementação lógica de um algoritmo ao descobrir que uma função ou um componente externo "*já resolve o problema*", essa violação também constitui uma necessidade de *feedback*, devendo o aprendiz ser orientado à correta construção algorítmica da solução, já que ela poderia ser justamente o objeto de aprendizagem para aquele momento da disciplina;
- **Lembretes sobre as exigências da atividade:** Semelhantemente ao tópico anterior, é comum que o professor solicite que o aluno realize uma certa validação lógica na entrada de dados ou a implementação de determinado cálculo ou mesmo o uso de dado recurso da linguagem, por exemplo, uma função nativa mais performática. Para o aprendiz essas exigências podem passar despercebidamente e apenas durante a apreciação da atividade pelo professor é que um *feedback* será oferecido juntamente com a nota da atividade. Nossa proposta é a integração de assistentes de fornecimento do *feedback* de exigências durante a realização das atividades. O ambiente deverá fornecer as condições para que o professor especifique não somente os resultados desejáveis no comportamento do programa, mas além disso, deve permitir que sejam especificadas as exigências de uso de certos recursos da linguagem ou da necessidade implementação de trechos de códigos requeridos a completude da solução;
- **Comentários sobre testes falhos da solução:** Os assistentes para comentários em testes falhos é um recurso do *AmPaRe* para o

feedback personalizado conforme o tipo de falha identificado. Nossa abordagem para esse recurso parte do princípio que comumente o aluno resolve parcialmente um problema, enquanto que o programa executa corretamente para alguns casos, para certos valores de entrada onde o sistema falha. Muitos desses casos são de conhecimento do professor devido sua vivência e a sua experiência na disciplina, tais falhas são por muitas vezes esperadas pelo docente e torna-se uma oportunidade do professor apoiar os estudantes com *feedbacks* ajustados para estes casos de falhas, propiciando assim a reflexão e o aprendizado a cada ocorrência.

5.5.1. O Suporte a Autorregulação

Dentre os modelos de autorregulação disponíveis, Zimmermann (1998, apud GANDA e BORUCHOVITCH, 2018) formulou que o processo de autorregulação acadêmica envolve três fases. A primeira fase é anterior ao processo de aprendizagem e é onde se faz o planejamento da atividade, a segunda acontece durante a execução da atividade e abrange as variáveis que afetam a atenção e a ação do aprendiz. Na terceira fase é realizada a autoavaliação, na qual a pessoa procura refletir sobre o seu desempenho ao longo do processo e reage diante dos resultados obtidos. Essa última fase influencia diretamente no modo como o indivíduo se engajará nas atividades acadêmicas semelhantes no futuro, mostrando que há uma relação dinâmica e cíclica em todo aprendizado. Os seguintes recursos abaixo foram previstos visando a autorregulação:

- **Promovendo o planejamento da atividade:**
    1. O aluno deve possui acesso claro aos critérios que serão avaliados e aos conhecimentos prévios necessários para que ele elabore a atividade. Ao cadastrar uma atividade o professor deve elencar esses dois grupos, dessa forma o aprendiz poderá voltar a algum conteúdo e somente após ganhar confiança ele decidirá iniciar essa atividade. Não obstante, o aluno consciente de suas deficiências poderá realizar consultas a fim de obter a ajuda dentro de suas fraquezas reconhecidas;
    2. O aluno poderá realizar uma pesquisa com bases nos critérios da atividade atual e assim obter outras atividades análogas onde ele

tenha desempenhado as mesmas habilidades. Para Júnior, Fechine e Costa (2009) é salutar ajudar o aluno iniciante a refletir sobre os aspectos que tornam tais problemas semelhantes;

- **Promovendo a auto experimentação:** É disponibilizado durante todo tempo uma área de codificação apartada, que possibilitará ao aprendiz experimentar os recursos da linguagem ou trechos de códigos, sem que isto venha interferir diretamente na sua atividade. Após avaliar um dado comportamento experimentado o aluno decidirá se incorpora ou não esse recurso na sua solução do problema;
- **Promovendo a autoavaliação:** Ao receber o resultado da avaliação o professor poderá fornecer os comentários para cada critério e habilidade avaliada. Além da nota final numa atividade, é interessante oportunizar que o estudante avalie sua performance em cada habilidade exigida, isto permitirá que o aprendiz realize as reflexões que poderão promover as suas condutas nas próximas atividades. Neste tempo o aluno deverá ser capaz de registrar quais são as suas considerações e sua autoavaliação na atividade. Tais recursos permitirão que o aprendiz externalize sua reflexão e que professor realize sua autorreflexão enquanto educador-educando ao avaliar o retorno deste aluno.

5.5.2. O Ambiente de Resolução e de Versionamento de Código

A proposta é que o ambiente propicie os recursos necessários para que o desenvolvimento de código-fonte da solução seja no mesmo local onde se encontra o repositório das atividades. No mais, esse ambiente deverá ser acessível na web de forma que o aluno não tenha dificuldades com a instalação e com a configuração do ambiente de programação e, portanto, consiga utilizá-lo fora dos momentos em laboratório e em qualquer lugar.

Dentre as características previstas no ambiente de resolução, incluem:

- **A criação de código com destaque nos comandos da linguagem:** Este é um recurso simples, porém importante por apoiar o aluno na leitura e navegação no seu código;

- **Autoedentação de código:** Esse recurso permite que o código seja auto organizado à medida que o aluno vai digitando;
- **Auto complementação de chaves, colchetes, parênteses e aspas:** o autocomplemento é um recurso que apoia o aluno durante a realização das atividades e atenua as ocorrências de erros de sintaxe;
- **Autossalvamento dos dados e versionamento do código:** O autossalvamento ocorre em todo momento que o aluno executa o programa, seja para testes ou ao submetê-lo ao professor. Ao salvar, o estudante poderá nomear a versão atual do código e acessá-la a qualquer momento. O versionamento de código-fonte permite ao aluno restaurar uma versão anterior ou que ele consulte alguma estratégia experimentada numa versão antiga do programa;
- **Compilador/interpretador e execução online:** Esse recurso dará as condições de portabilidade e mobilidade ao aluno, permitindo que ele continue suas atividades fora do laboratório, por exemplo em casa, na biblioteca ou em qualquer outro local com acesso à Internet.

5.6. A Arquitetura Proposta

A arquitetura proposta é realizada através da junção do padrão *MVC* e da arquitetura orientada a serviços (SOA), tais recursos foram utilizados com o objetivo de separar logicamente as responsabilidades do ambiente, por permitir sua extensibilidade e por proporcionar um baixo acoplamento entre os componentes do sistema.

Conforme apresentado no Capítulo 3, o padrão *MVC* (*Model*, *View*, *Controller)* separa os elementos de um sistema permitindo mudá-los de forma independente. Por exemplo, pode-se adicionar uma nova visão ou alterar uma exibição existente sem quaisquer necessidades de alterações nos dados subjacentes do modelo. Além do mais, uma arquitetura em camadas é mutável e portável, pois enquanto sua interface estiver inalterada uma camada poderá ser substituída por outra equivalente e mesmo que uma camada mude ou tenha novos recursos adicionados apenas a camada adjacente será afetada. Na Figura 19 é ilustrada uma visão geral da organização das camadas propostas para o *AmPaRe*.

Se comparado ao modelo *MVC* padrão, o modelo apresentado na Figura 19 possui adicionalmente a camada *Persistência* e a camada *Serviços*. A camada *Persistência* visa isolar a responsabilidade de persistência dos dados e oferecer os recursos necessários para manutenção deles, abstraindo esses detalhes das camadas superiores que apenas precisarão conhecer a camada *Negócio*.

Já a camada *Serviços* (Figura 19) será responsável por disponibilizar os recursos externos, tais como os agentes inteligentes criados fora da aplicação e que poderão ser anexados ao ambiente proposto. Propõe-se que os assistentes desenvolvidos externamente adotem os padrões definidos pela arquitetura orientada a serviços (SOA), essa abordagem promoverá a colaboração e a extensão do ambiente, possibilitando que os assistentes sejam sistemas distribuídos e criados com recursos próprios de memória e de processamento, além disso a abordagem SOA permitirá que esses assistentes evoluam independentemente do restante do sistema e que novos assistentes sejam integrados ao ambiente de forma mais flexível.

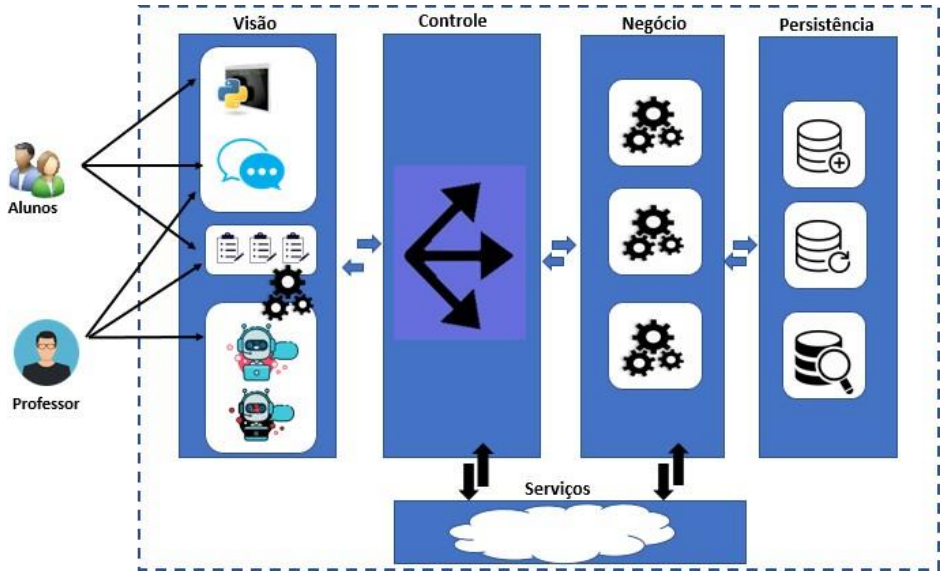


Figura 20 - Arquitetura em camadas do ambiente proposto.

Fonte: Autoria Própria

5.6.1. Camada de Visão

De acordo com o padrão *MVC*, a camada de visão será o componente que definirá a forma como os dados serão apresentados ao usuário.

Em se tratando de uma solução web, a camada de visão será responsável por renderizar o conteúdo *HTML*, as imagens, os estilos dentre outros aspectos ligados a interface com o usuário. Além disso, essa camada deverá revelar os eventos nos botões disponíveis em tela, comunicando-os à camada *Controle*, assim como os eventos acionados a partir de formulários relacionados a atualização de dados em tela.

Destaca-se nesta camada a visualização do ambiente de desenvolvimento, onde o estudante poderá codificar as soluções das atividades propostas inteiramente no *Browser* sem a necessidade de instalação de componentes ou realizar a conversação sobre as suas dificuldades.

5.6.2. Camada de Controle

A camada *Controle* será responsável pelo mapeamento das ações do usuário sob o modelo, por rotear as ações para a camada *Negocio* e também por selecionar as visões adequadas para serem enviadas para renderização no *Browser*.

Um importante objeto instanciado nesta camada é oriundo da classe *CoordenadorDeAssistencia* (Figura 20) que será responsável por comunicar um pedido de suporte a todos assistentes e por associar um certo assistente à conversação com o aluno. A cada demanda de atendimento os assistentes serão requisitados e suas ações deverão ser intercambiadas para camada *Visão* através da camada *Controle*.

5.6.3. Camada de Negócio

A camada *Negócio* implementará as classes com seus os atributos e os métodos conforme padrão POO. O diagrama de classe ilustrado na Figura 20 apresenta essas classes e a relação entre elas.

As interações com os assistentes ocorrem através das relações com as classes *ContextoFacade*, *CoordenadorDeAssistencia* e *AnalisadorIntencaoConversacao* e das realizações oriundas das interfaces *IAssistenteBase* e *IAssistenteMediator*.

A interface *IAssistenteBase* definirá a estrutura básica de um assistente e seus comportamentos. Diversos tipos de assistentes poderão ser implementados através da realização da interface *IAssistenteBase* ao tempo que os objetos dessas classes adquirirão a compatibilidade com as demais classes que conhecem a interface *IAssistenteBase*.

As classes *AssistenteDeErro*, a *AssistenteConversacional* e a classe *AssistenteRemoteProxy* são realizações da interface *IAssistenteBase*. Já as classes *AssistenteDeErro* e *AssistenteConversacional* permitirão a codificação direta de dois tipos de assistentes: o conversacional e o de análise de código, enquanto que a classe *AssistenteRemoteProxy* permitirá que sejam anexados assistentes externos no ambiente através da integração das requisições com assistentes implementados com a SOA.

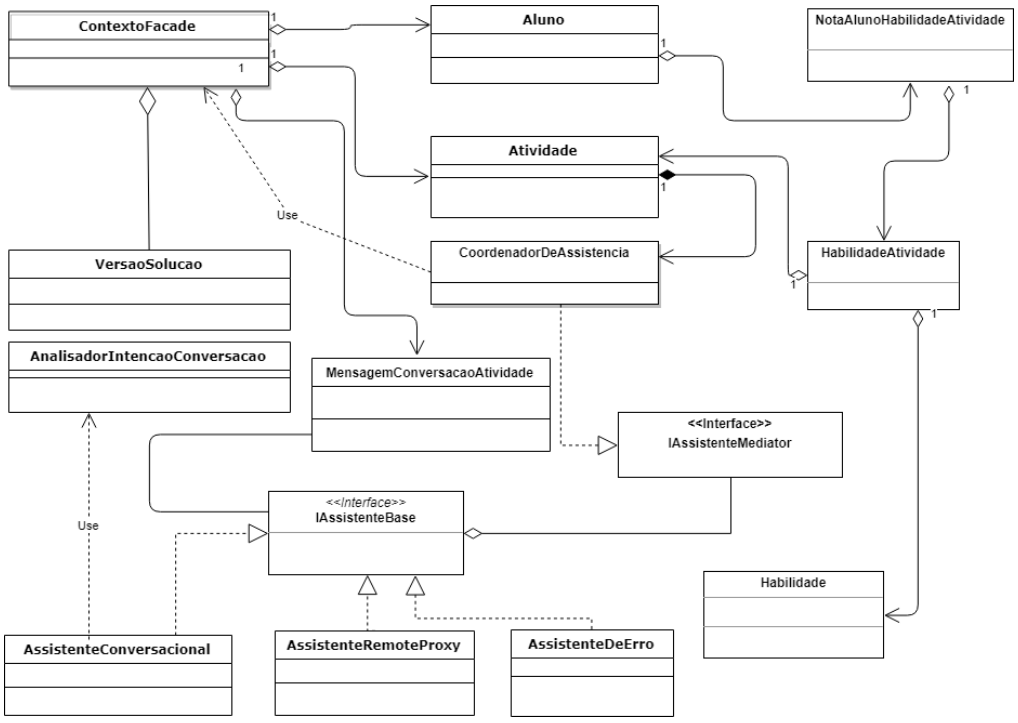


Figura 21 - Diagrama de classes do ambiente proposto.

Fonte: Autoria Própria

O *Design Pattern Facade* (GAMMA, 2011) foi incorporado na classe *ContextoFacade*, sendo que esta classe será o principal recurso utilizado para representar o subgrupo do sistema responsável por agregar os diversos objetos

ligados ao contexto da conversação, ao armazenamento das versões de código e ao repasse das demandas de ajuda para a instância da classe *CoordenadorDeAssistencia*.

A classe *AnalisadorIntencaoConversacao* visa oferecer os recursos de classificação de texto na busca de padrões relacionados à intenção do aluno ao fornecer uma mensagem de solicitação de suporte, muito embora outros recursos possam ser empregados internamente nos ACs ao avaliar a conversação com o aprendiz.

O *Design Pattern Mediator* (GAMMA, 2011) deverá ser aplicado na classe *CoordenadorDeAssistencia* que então deverá propagar a solicitação da demanda de ajuda para todos os assistentes disponíveis para a atividade. Cada assistente vinculado à atividade comunica a mudança de seu estado ao *CoordenadorDeAssistencia* que em seguida recepciona as condições de cada assistente em relação a demanda ocorrida.

5.6.4. Camada de Persistência

A camada de persistência será responsável por permitir que os objetos da aplicação sejam armazenados, consultados, atualizados ou excluídos na base de dados.

5.6.5. Camada de Serviços

A camada de *Serviços* oferecerá os recursos externos que poderão ser integrados ao *AmPaRe*. Uma possibilidade prevista é a utilização dos agentes inteligentes (RUSSEL; NORVIG, 2013) implementados como serviços em SOA cumprindo o papel de assistentes conversacionais ou de análise de código.

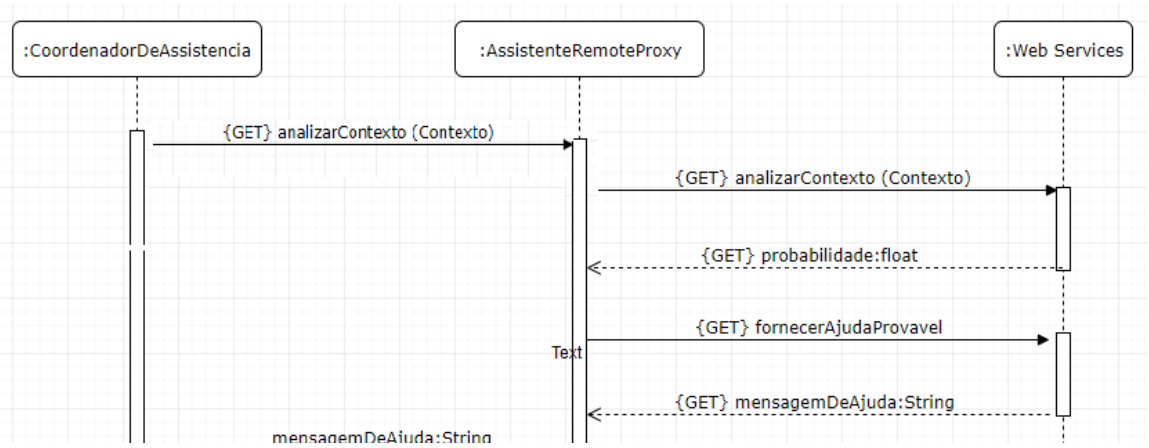


Figura 22 - Diagrama de sequência de consulta ao web services.

Fonte: Autoria Própria

Na classe *AssistenteRemoteProxy* (Figura 21), uma outra implementação da interface *IAssistenteBase*, deverá implementado o *Design Pattern Proxy* (GAMMA, 2011) para abstrair o acoplamento de um assistente remoto e residente em *Web Services* (SOMMERVILLE, 2011).

Análogo ao comportamento dos demais assistentes, o assistente remoto deverá ser capaz de receber o contexto de uma demanda de suporte e apontar sua condição para o atendimento dessa demanda. O uso desses assistentes deve ser transparente para o restante da aplicação, embora eles estejam implementados fora da aplicação. A Figura 21 ilustra o digrama de sequência com a mensagens de solicitação e de retorno dos assistentes externos ao se comunicar com a aplicação através da classe *AssistenteRemoteProxy*.

5.7. Considerações Finais do Capítulo

A proposta apresentada buscou oferecer os recursos para o atendimento das demandas dos alunos durante a realização das atividades de programação, de forma que esse suporte fosse obtido no mesmo ambiente onde o aprendiz desenvolve as suas atividades. Buscou-se preservar a autonomia do professor em deliberar qual os tipos de necessidades ficariam por conta dos assistentes disponíveis. Além disso, a solução proposta prevê que ao longo do tempo cada assistente ganhe o estilo e a forma de mediação do professor, preservando assim uma coerência entre as mediações realizadas pelo professor e aquelas fornecidas pelos assistentes.

Não é esperado que os assistentes substituam o docente em todas as suas atividades de suporte e de *feedback* às dúvidas dos alunos, mas que esses assistentes venham liberar o professor das demandas recorrentes de atendimento, onde hajam ações mediadoras já previstas pelo professor e que ele tenha decidido transferir o atendimento para o ambiente.

É importante ressaltar que os assistentes do *AmPaRe* não se tratam de um Sistema Tutor Inteligente como descrito em Anderson, Boyle e Reiser (1985), Boulay (1986) e (UENO, 1989), visto que eles não apresentam todas as características

essenciais necessárias a estes sistemas, por exemplo, o módulo de estratégias pedagógicas, o módulo especialista ou mesmo um modelo robusto sobre o estudante.

Ressalta-se, portanto, que a proposta de ambiente aqui apresentada visa permitir que o professor especialize assistentes que agirão de forma autônoma. Todavia, é previsto que esses assistentes aprendam com o estilo do professor, enquanto que o docente estabelece o campo de atuação de cada um destes assistentes.

6. Tecnologias de Suporte

Neste Capítulo serão abordadas as tecnologias de suporte a implementação do ambiente proposto.

Na Seção 6.1 será apresentada a ferramenta *Skulpt* utilizada na implementação do ambiente de codificação pelo qual será possível criar um editor de código online e rodar os programas dos alunos diretamente no navegador de Internet.

Em seguida, na Seção 6.2 serão apresentadas as ferramentas disponíveis no pacote *Scikit-Learn* da qual serão utilizados os recursos de aprendizagem de máquina dedicados a classificação de texto.

Na Seção 6.3 serão apresentados os algoritmos que serão utilizados no pré-processamento de linguagem natural. Logo depois, na Seção 6.4 serão abordados os aspectos ligados ao *Pylint*, uma ferramenta dedicada a análise de códigos desenvolvidos com a linguagem de programação *Python*.

Por último, na Seção 6.5 será apresentado o micro framework *Flask* que será utilizado na implementação do ambiente web e na separação da solução em camadas.

6.1. A Ferramenta Skulpt

A ferramenta *Skulpt* é uma iniciativa de código-fonte aberto idealizada por Scott Graham que tem por objetivo disponibilizar um interpretador da linguagem *Python* rodando inteiramente na web (SKULPT, 2018). O Skulpt foi desenvolvido a partir da linguagem *Javascript* e roda diretamente no lado do cliente, ou seja, através do navegador de Internet, sem a necessidade de instalações no lado do servidor ou de outros componentes na máquina do usuário.

A Skulpt é uma alternativa a outras soluções tais como o *Brython*[2], *Pyodide*[3] e o *IronPython*[4]. Sua escolha se deu por sua rápida integração a um projeto e pela

[2] https://brython.info/
[3] https://github.com/iodide-project/pyodide
[4] https://ironpython.net/

compatibilidade dessa ferramenta com as versões 2.x e as atuais versões 3.x do *Python*, e também por sua performance ao interpretar e rodar os programas.

Os arquivos *ast.js*, *parser.js*, s*ymtable.js*, *compile.js* e *tokenize.js* são os responsáveis pela transformação de código Python num equivalente em *JavaScript*.

6.2. O Pacote Scikit-Learn

O pacote *Scikit-Learn* (PEDREGOSA et al., 2011) é uma biblioteca de código-fonte aberto em *Python* que reúne a implementação dos variados algoritmos relacionados à aprendizagem de máquina. As implementações disponíveis no *Scikit-Learn* são de alta performance e incluem algoritmos clássicos para a classificação de documentos, raciocínio probabilísticos, processamento de linguagem natural e redes neurais dentre outros, além de ferramentas amplamente utilizadas para a avaliação de modelos, preparação de dados e procedimentos pré-processamento de dados (BUITINCK et al., 2013).

As vantagens identificadas no *Scikit-Learn* são que i) ele é distribuído sob a licença BSD; ii) ele incorpora um código compilado para eficiência, diferentemente outras soluções; ii) depende apenas de *numpy* e *scipy*, o que torna sua distribuição facilitada (PEDREGOSA et al., 2011).

6.2.1. Multinomial Naive Bayes

A classe MultinomialNB disponível no pacote *sklearn.naive_bayes* implementa um classificador Naive Bayes Multinomial que é adequado para classificação documento que possuem características discretas (SKLEARN, 2019), por exemplo, a classificação de texto baseada na frequência de ocorrências dos termos.

Complementarmente ao processo do MNB, foi utilizada a classe *CountVectorizer* disponível no pacote *sklearn.feature_extraction.text*. Esta classe oferece os recursos para a conversão de uma coleção de documentos de texto numa matriz de contagens de termos.

6.3. O NLTK - Natural Language Toolkit

O pacote *NLTK* (*Natural Language Toolkit*) é uma biblioteca de ferramentas desenvolvidas em Python e usada para auxiliar linguistas, engenheiros, estudantes, educadores e pesquisadores da área de processamento de linguagem natural (BIRD; KLEIN; LOPER, 2009).

De acordo com Loper e Bird (2002) a principal vantagem do NLTK dá-se por conta de sua simplicidade de uso, sua vasta documentação disponível para cada recurso desenvolvido, pela consistência oferecida nos resultados, por sua extensibilidade e pela forma modular de cada recurso oferecido.

No contexto dessa dissertação pretendeu-se utilizar os recursos da NLTK para o pré-processamento das sentenças enunciadas nas dificuldades registradas pelos alunos no ambiente de conversação. Portanto, foram identificadas as ferramentas *RSLP Stemmer* e *StopWords*, respectivamente responsáveis pela redução das palavras flexionadas (*stemização*) e pela remoção dos termos de pouca relevância no significado do texto.

6.3.1 Ferramenta RSLP Stemmer

A ferramenta o RSLP é um acrônimo para Removedor de Sufixos da Língua Portuguesa e foi proposto por Orengo e Huyck (2001) no tratamento de stemização de sentenças na língua portuguesa.

De acordo como Coelho (2007) o algoritmo RSLP foi submetido a vários testes de desempenho e mantém-se superior a outras alternativas. Diversos trabalhos recentes com o processamento de linguagem natural envolveram o uso desse algoritmo, assim como utilizado em Panceri e Menezes (2015).

6.3.2. A Ferramenta StopWords

A *StopWords* disponível no pacote *nltk.corpus* implementa o algoritmo para remoção de palavras com baixo valor semântico no texto, nisto incluem os artigos, preposições e conjunções.

6.4. A Ferramenta PyLint

O *Pylint* é uma ferramenta que verifica os erros no código desenvolvido na linguagem de programação Python. O Pylint tenta impor um padrão de codificação e procura por códigos mal escritos (PYLINT, 2019). Ele também pode procurar por determinados erros de acordo com os tipos especificados, pode recomendar a reescrita de determinados trechos de código e oferece detalhes sobre códigos escritos com uma complexidade desnecessária.

Conforme a documentação técnica do Pylint as duas principais classes dessa ferramenta são a *pylint.lint.Run* e a *pylint.lint.PyLinter* (PYLINT, 2018).

A classe *pylint.lint.Run* disponibiliza o objeto responsável por iniciar o *Pylint*. Este objeto recebe os parâmetros necessários para inicialização do ambiente de análise do código e também realiza o carregamento dos plugins especificados na inicialização. Em seguida, ele cria uma instância de *pylint.lint.PyLinter* (PYLINT, 2018).

Já a classe *pylint.lint.PyLinter* é responsável por coordenar o processo de lintagem, ou seja, ela analisa o código em busca eventos de erros e outras ocorrências definidas para a análise conforme os parâmetros de inicialização (PYLINT, 2018). O *PyLinter* analisa a configuração e a fornece para os verificadores e também para os plugins carregados, em seguida, ela lida com as mensagens emitidas pelos verificadores, lida com os relatórios de saída e executa os verificadores de código.

6.5. Servidor web Flask

O *Flask* é um micro framework para uso em aplicações web desenvolvidas em Python. Esse framework fornece as funcionalidades básicas necessários à implementação de web services e aplicações web com o MVC (LOKHANDE et al., 2015). O Flask objetiva ser simples, todavia possibilita a integração com vários plugins estendendo assim as suas funcionalidades.

Algumas características são relevantes para a escolha do Flask na construção de uma aplicação web, são:

- Suporta testes unitários;
- Usa os modelos Jinja2, uma é uma linguagem de templates moderna e amigável ao designer modelada e otimizada a partir dos modelos do Django10;
- Suporte para cookies seguros;
- Extensa documentação;
- Compatibilidade com o *Google App Engine*;
- Baseado em Unicode;

Uma aplicação web em *Flask* segue normalmente uma estrutura de arquivos e diretórios que permite separar as responsabilidades do sistema.

6.6. Ferramenta SQLAlchemy

O *SQLAlchemy* é um kit de ferramentas para mapeamento objeto-relacional (ORM) na linguagem de programação Python e é integrável ao micro framework Flask. O SQLAlchemy fornece um conjunto completo de padrões de persistência de níveis corporativos bem conhecidos e é projetado para o acesso a banco de dados de forma eficiente e com alto desempenho (MYERS; COPELAND, 2015).

O *SQLAlchemy* apresenta um meio de associar classes do Python em tabelas de banco de dados e as instâncias dessas classes (objetos) como linhas nestas tabelas correspondentes. Ele inclui um sistema que sincroniza de forma transparente todas as mudanças no estado entre objetos e suas linhas relacionadas, chamadas de processos, bem como um sistema para expressar consultas de banco de dados em termos de classes definidas pelo usuário e seus relacionamentos definidos entre si (SQLALCHEMY, 2018).

Atualmente o SQLAlchemy suporta a persistência para os principais bancos de dados relacionais de mercado, dentre eles o SQLite, Postgresql, MySQL, Oracle, MS-SQL, Firebird e o Sybase.

7. Prova de Conceito

Neste Capítulo será apresentada um protótipo do ambiente proposto.

Na Seção 7.1 será apresentada uma visão geral da implementação, o escopo atendido e as tecnologias envolvidas no arcabouço geral do ambiente.

Na Seção 7.2 serão apresentados os casos de uso implementados para o ambiente do professor, nos quais envolvem as funcionalidades ligadas ao gerenciamento das atividades, a seleção de assistentes para o acompanhamento dessas atividades e os recursos ligados ao tratamento das dificuldades não atendidas pelos assistentes. Também nesta Seção serão apresentados os recursos implementados para que o docente crie e especialize as instâncias dos assistentes analisadores de código e os conversacionais.

Em seguida, na Seção 7.3 será apresentado o ambiente sob a perspectiva de acesso pelo aluno, sendo apresentadas as funcionalidades de gestão das atividades disponíveis, o editor online onde o aluno poderá codificar e executar seus programas, o ambiente de conversação onde será possível que o aluno registre as suas dificuldades e interaja com os assistentes disponíveis.

Na Seção 7.4 serão apresentados os detalhes ligados a implementação do assistente conversacional dentro do escopo desta prova de conceito.

Em seguida, na Seção 7.5 serão apresentados os aspectos ligados a implementação dos assistentes analisadores de código que nesta prova de conceito serão responsáveis pelas assistências nos casos de erros identificados no código-fonte.

Por último, na Seção 7.6 serão apresentadas as conclusões finais deste capítulo.

7.1. Visão Geral da Implementação

Na condução desta prova de conceito foram implementadas as funcionalidades para que professor crie as atividades e especialize as instâncias dos assistentes responsáveis pelo acompanhamento dos alunos durante estas atividades. Aos alunos foram disponibilizados o ambiente para o acesso as atividades, o editor online para resolução das atividades e o ambiente de conversação onde o aluno poderá enunciar as suas dificuldades e então obter o feedback.

A linguagem de programação suportada no ambiente de resolução de problemas foi o *Python* (PYTHON, 2018) na versão 3.7 devido a sua sintaxe simples e a facilidade desta linguagem na escrita de instruções, não sendo necessário muitos comandos para construir um programa (GRANDELL et al., 2006). A linguagem Python é utilizada por diferentes níveis de programadores, foi inicialmente criada para facilitar o aprendizado de programação, embora também seja utilizada em diversos ambientes comerciais e de pesquisa. Outro diferencial desta linguagem é que existe uma grande comunidade de usuários que compartilham seus aprendizados e disponibilizam uma enorme quantidade de materiais sobre a linguagem na Internet (GRANDELL et al., 2006; JENKINS, 2004). Todavia, o modelo conceitual estabelecido poderá ser implementado para atendimento a outras linguagens de programação ou mesmo a construção de um ambiente com suporte a múltiplas linguagens, neste caso o maior impacto seria na construção de assistentes analisadores de código (ACOs) específicos para cada linguagem ou construí-los com a capacidade de identificar e lidar com cada tipo de linguagem suportada.

Para esta prova de conceito o escopo implementado compreendeu a criação de dois tipos de assistentes, o assistente de erros, sendo um tipo de ACOs e o assistente conversacional.

O assistente de erros realiza o suporte no entendimento das causas de ocorridos durante a tentativa de execução do programa ou identificados após o aluno salvar ou submeter o código desenvolvido. Já o assistente o conversacional é responsável por receber e tratar as dificuldades informadas pelo aluno através da área de conversação via texto disponível no mesmo ambiente de resolução de problemas.

Buscou-se implementação de um ambiente inteiramente funcional via web a fim de permitir que tanto professor quanto alunos acessem o ambiente de qualquer lugar e a qualquer tempo. Tal iniciativa propicia uma extensão do laboratório, além do mais viabiliza a economia de tempo gasto com a preparação do ambiente de codificação necessário à resolução das atividades propostas.

O navegador homologado para este protótipo foi *Google Chrome* versão 74.0.3, muito embora não haja nenhuma restrição a outros navegadores devidamente atualizados para suporte a *JavaScript V8, HTML 5* e *CSS3*.

A Figura 22 ilustra uma visão geral da implementação sob a perspectiva do professor ao acompanhar uma atividade proposta e do aluno o tentar resolver uma

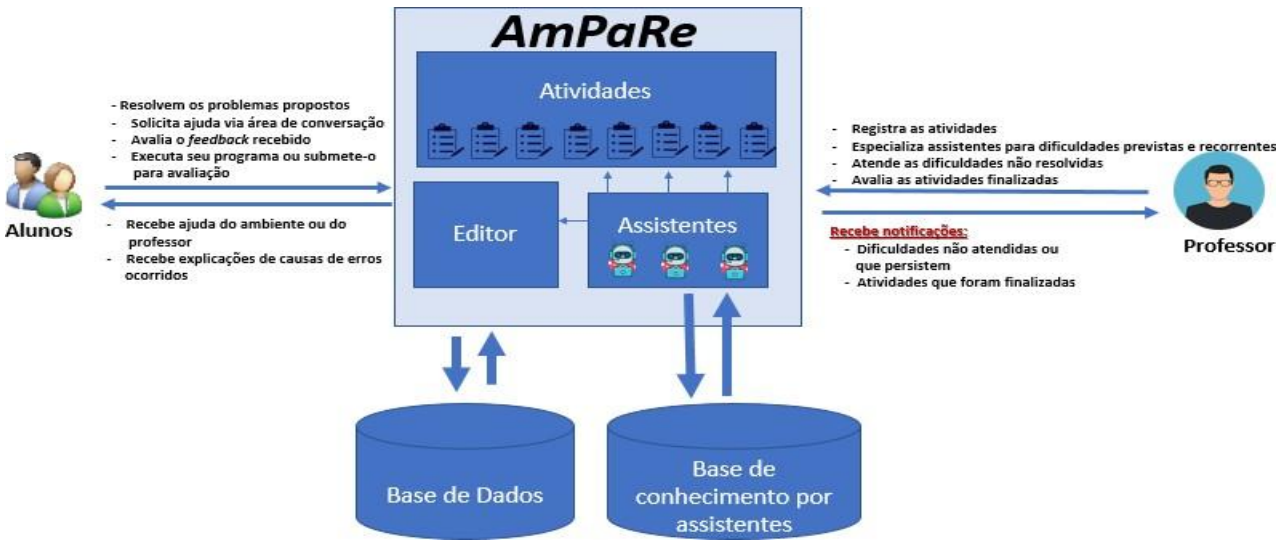


Figura 23 - Visão geral do ambiente desenvolvido.
Fonte: Autoria Própria

atividade.

O professor acessa o ambiente para criar instâncias dos assistentes disponíveis, repassando para esses agentes as dificuldades esperadas e a mensagem de feedback a ser enviada para aluno durante a atuação dessas instâncias. Para um mesmo tipo de assistente o professor poderá criar diversas instâncias nomeadas, cada qual atuando isoladamente a partir do treinamento inicial e da inteligência desenvolvida durante o fornecimento de suporte. Ao criar as

atividades o professor escolherá quais dessas instâncias ficarão responsáveis pelo monitoramento das dificuldades e suporte.

Já o aluno acessa o ambiente e a partir do repositório de atividades ele escolhe uma tarefa para resolvê-la. Em seguida o editor de *Python* é liberado e a seleção de assistentes da atividade o acompanham a fim de propiciar o suporte nas ocorrências das dificuldades previstas. Ao receber uma ajuda o aluno poderá avaliar o *feedback* recebido, caso essa ajuda seja mal avaliada, a demanda será encaminhada para o professor que então poderá complementar algum *feedback* fornecido ou apoiar o aluno presencialmente se a gravidade da dificuldade assim o exigir.

Durante o atendimento das demandas não solucionadas pelos assistentes o professor poderá responder ao aluno e ao mesmo tempo aproveitar o momento para especializar uma instância de assistente mais adequada para aquela demanda ou para outras semelhantes numa eventual recorrência.

A Figura 23 apresenta a tela de entrada da aplicação, acessível ao professor e aos alunos.

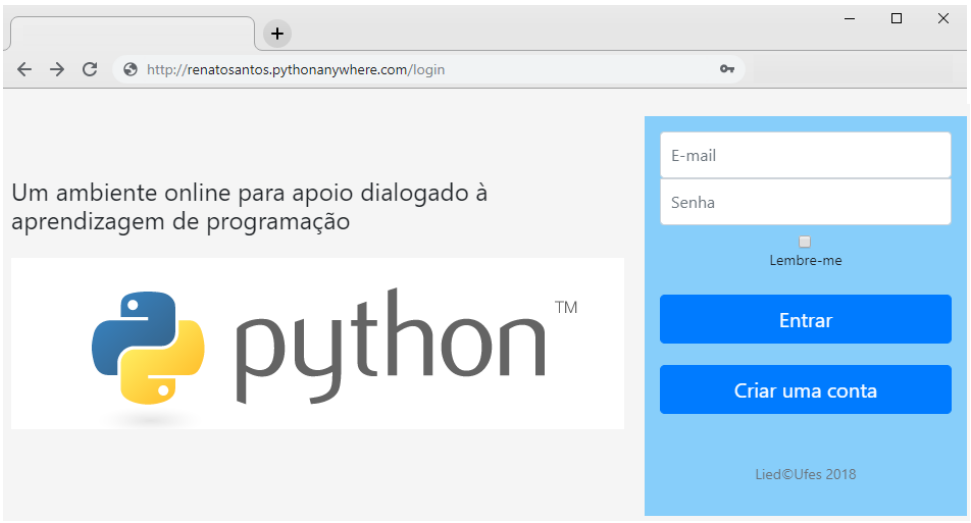


Figura 24 - Tela inicial de entrada no ambiente.

Fonte: Autoria Própria

Outros recursos foram disponibilizados no ambiente de resolução, seguem:

- **Console Python online:** Este recurso permite que onde aluno teste comandos ou experimente uma estratégia de programação sem que seja de comprometido o código-fonte da atividade em desenvolvimento;
- **Área de execução do programa:** ao pressionar o botão "*Salvar e Executar*" do editor uma janela é sobreposta ao ambiente e então a execução é exibida com o comportamento do programa ou com os erros do interpretador exibidos.

7.2. O Ambiente do Professor

A fim de ilustrar rapidamente as funcionalidades implementadas no ambiente do professor foi criado o diagrama de casos de uso da Figura 24 e as desenvolvidas serão apresentadas no decorrer desta seção.

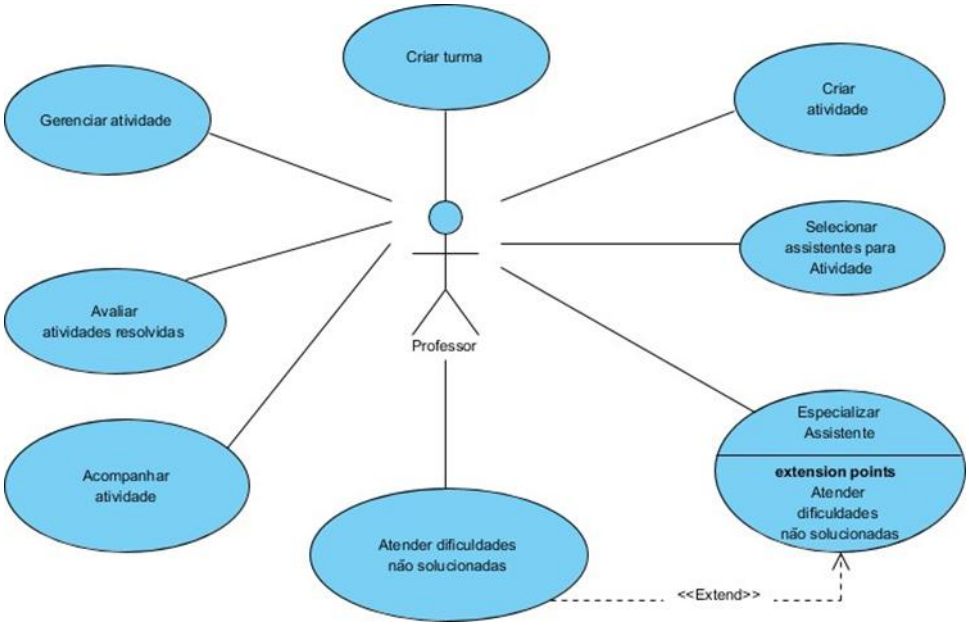


Figura 25 - Casos de usos das funcionalidades implementadas para o professor.
Fonte: Autoria Própria

7.2.1. Gerenciamento das Atividades

Através da tela Gerenciar Atividades são disponibilizadas as funcionalidades que atendem os seguintes casos de uso: *“Criar atividade”, “Gerenciar Atividade”, “Avaliar*

Atividade” e “*Selecionar assistentes para a atividade”*. A Figura 25 apresenta a visão do professor ao gerenciar o repositório das atividades no ambiente.

Ao clicar no botão "*Nova Atividade*" (Figura 25) é disponibilizado a área onde o professor especifica o enunciado, o título, decide a marcação do campo “*Disponibilizar para ao aluno?* ”, em seguida preenche o período de validade da atividade. Uma atividade somente será liberada para o aluno dentro do período indicado pelo professor e se marcada a opção “*Disponibilizar para ao aluno?* ”.

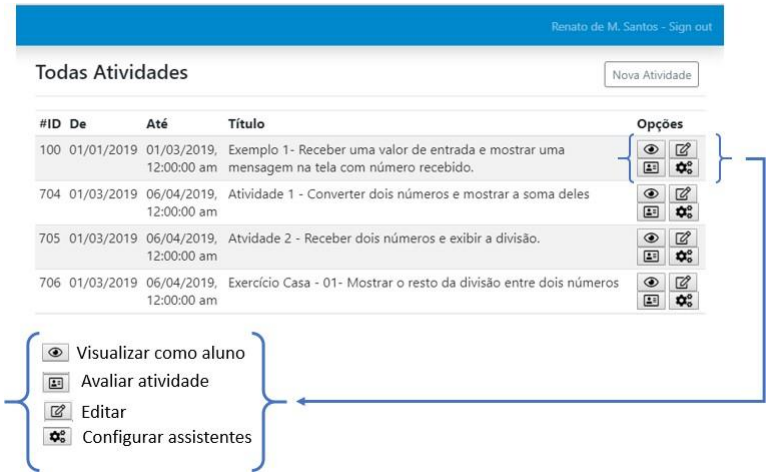


Figura 26 - Gerenciamento das atividades (visão do professor)

Fonte: Autoria Própria

No momento que uma atividade é criada a tela do professor é imediatamente redirecionada para a tela “*Habilidades Avaliadas*” onde o professor poderá especificar quais são as habilidades requeridas para aquelas atividades. Na seleção das habilidades o professor especifica a nota máxima correspondente a cada habilidade, sendo que o valor total da atividade corresponde ao somatório das notas de cada habilidade avaliada.

7.2.2. Seleção de Assistentes por Atividades

Partindo-se do princípio que o professor já tenha especializado uma comunidade de assistentes por tipos de dificuldades esperadas, ao acessar uma atividade o docente poderá clicar sobre o botão "*Configurar Assistentes*” (Figura 25) e então é liberado ao o professor a tela para adicionar ou remover assistentes da atividade em questão.

Múltiplas instâncias de assistentes devidamente nomeados são disponibilizadas para o professor selecioná-los para uma atividade (26).

Uma atividade poderá ter vários ou nenhum assistente vinculado, caso não haja ao menos um assistente adequado para o atendimento ao aluno é enviada a seguinte mensagem no ambiente de conversação: "*Não há assistentes disponíveis para essa dificuldade, encaminharei para o professor atendê-la!*".

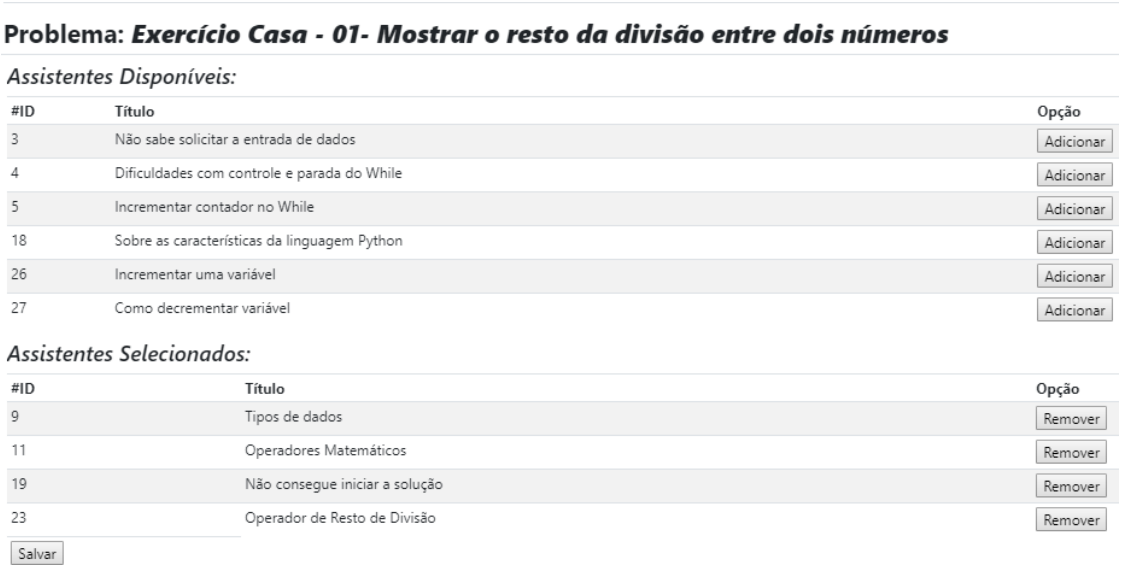


Figura 27 - Tela de seleção de assistentes por atividade.

Fonte: Autoria Própria.

7.2.3. O Tratamento das dificuldades não resolvidas pelos assistentes

Nesta subseção serão apresentadas as interfaces relacionadas aos casos de uso *“Especializar Assistente”* e o *“Atender as dificuldades não solucionadas”.*

Toda dúvida ou dificuldade não resolvida por um assistente ou mal avaliada por um aluno é encaminhada para o professor respondê-la, seja presencialmente ou no ambiente ao complementar o *feedback* fornecido pelo assistente.

Ao tratar uma dificuldade encaminhada (Figura 27), o professor também poderá selecionar um assistente mais adequado ao tratamento daquela dificuldade para uma eventual recorrência dela ou de casos semelhantes. Essa nova atribuição será considerada pelos mecanismos internos do assistente ao se adaptar à nova

dificuldade e então ele buscará atender não somente ao caso específico e também as variações possíveis.

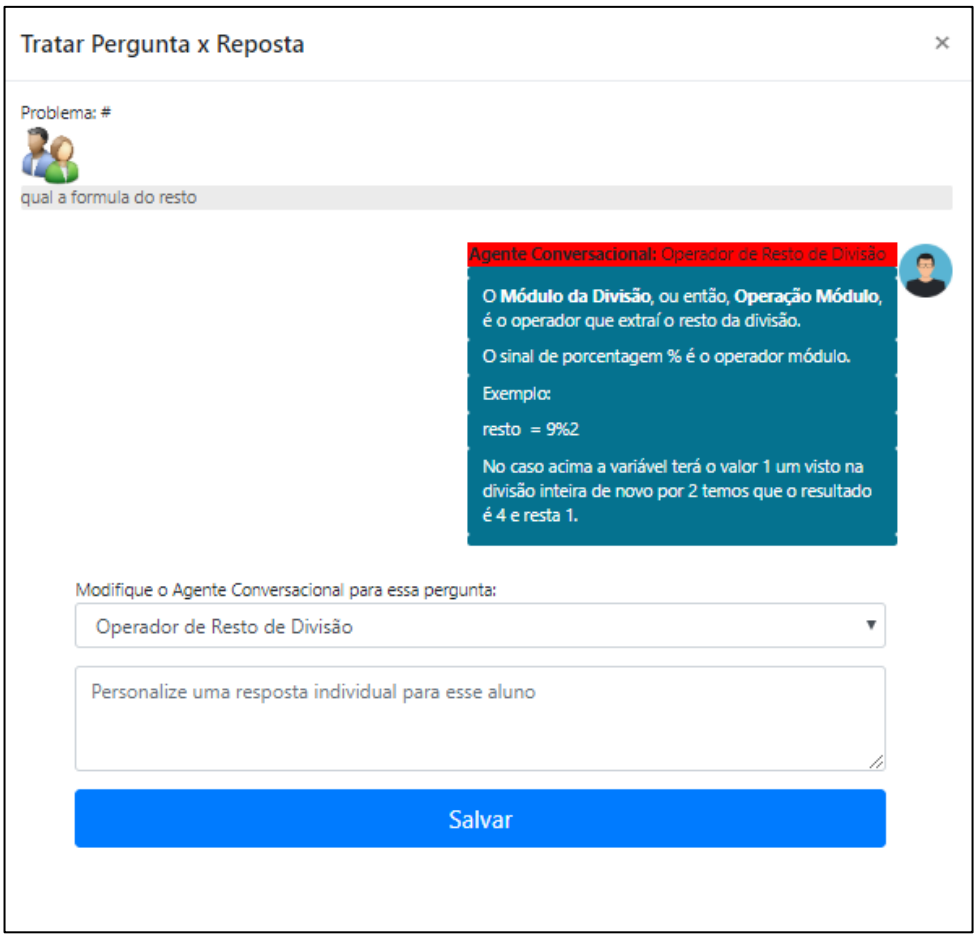


Figura 28 - Tratamento feedback mal avaliado.

Fonte: Autoria Própria

A especialização de um assistente durante a correção de dúvidas não atendidas foi implementada apenas para os assistentes conversacionais e o contexto associando enviado para esses assistentes representando uma nova demanda de atendimento envolveu apenas as dificuldades enunciadas pelos alunos a digitá-las no ambiente de conversação.

7.2.4. Gestão dos Assistentes Conversacionais

A criação de uma instância de um assistente conversacional é relativamente simples para o professor (Figura 28), bastando apenas que o docente informe o nome que

identificará o assistente, um apelido, um ícone que será utilizado representar o recurso durante a conversação e as mensagens fornecidas pelo assistente durante o diálogo.

A especialização uma instancia do assistente conversacional poderá ocorrer com base na experiência do professor, neste caso o docente poderá registrar exemplos de dúvidas já ocorridas e ação tomada. Essa atividade pode ser realizada logo que assistente é instanciado ou a qualquer momento clicando sobre o botão "*Treinar* “ (Figura 28).

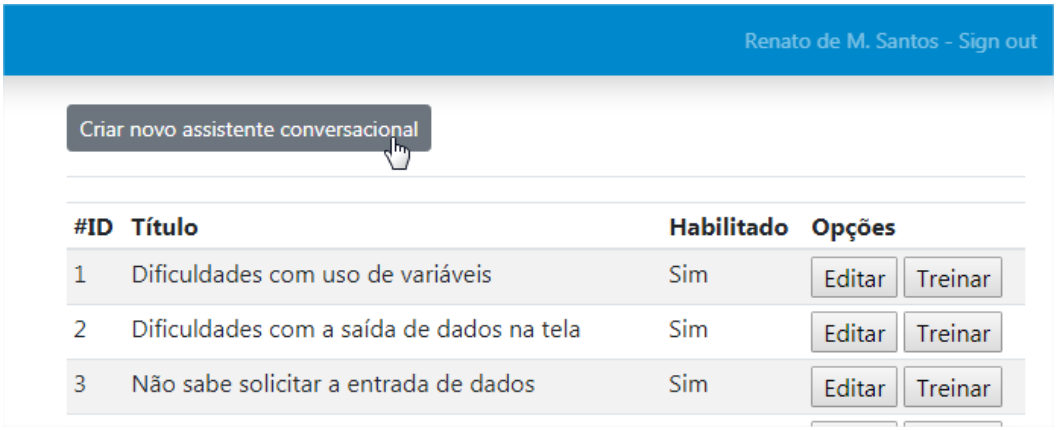


Figura 29 - Tela de gestão dos assistentes conversacionais.

Fonte: Autoria Própria

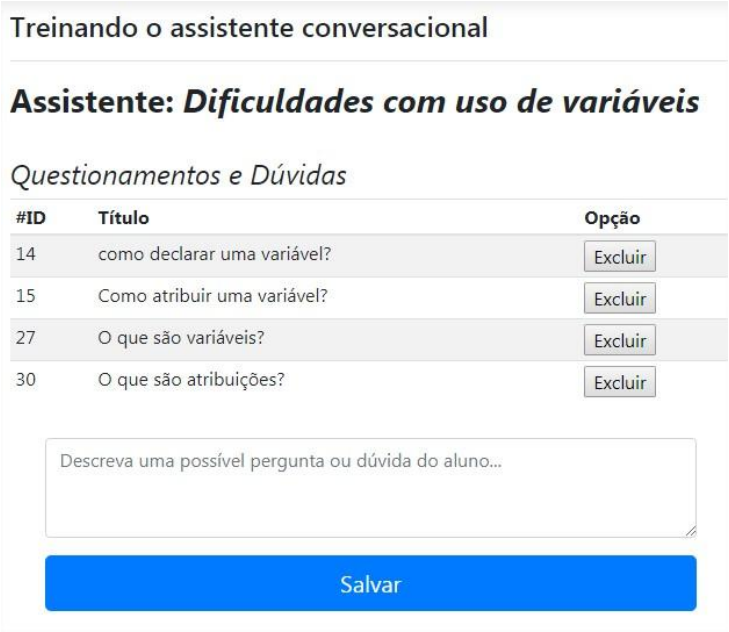


Figura 30 - Tela de treino dos assistentes conversacionais.

Fonte: Autoria Própria

O professor poderá registrar uma série de frases descrevendo as mais variadas formas de se expressar uma mesma dificuldade (Figura 29). Tal recurso permite que compartilhe a qualquer momento a sua bagagem de experiência sobre as possíveis formas do aluno expressar as suas dificuldades durante a conversação como os assistentes.

Ao ser alimentado com novas formas de perguntas, os assistentes utilizarão esses registros, bem como seu histórico interno de atendimentos bem-sucedidos para ser aperfeiçoar para os novos atendimentos semelhantes.

7.2.5. Gestão dos Assistentes para Erros Esperados

De forma análoga a criação de um assistente conversacional, ao criar uma instância do assistente de erro (Figura 30) o professor informa um nome, um apelido e um ícone correspondente a identidade visual desse assistente. A diferença é que para este tipo de assistente o professor escolhe um tipo de erro suportado pelo assistente e então registra a mensagem fornecida ao aluno.

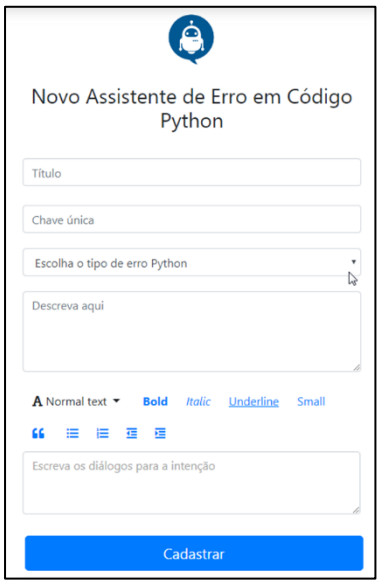


Figura 31 - Tela de criação de assistente de erro em códigos em Python.
Fonte: Autoria Própria

O assistente de erros foi implementado para o tratamento de erros informados pelo *Interpretador Python* e quando catalogados pela ferramenta *PyLint*. Portanto, ao criar uma dada instância de uma assistente de erros essa instância deverá ser associada a um erro catalogado. Ao analisar o código-fonte da atividade do aluno e identificar a incidência de erro no código, o assistente aciona o aluno através da área de conversação e envia *feedback* definido pelo professor para aquele contexto da atividade.

Salienta-se que diferentes instâncias dos assistentes de erros podem ser criadas para o mesmo tipo de erro de *Python*, dessa forma o professor poderá especializar distintos assistentes para um mesmo evento, fornecendo em cada atendimento mais ou menos detalhes conforme atividade que o assistente foi vinculado.

7.3. O Ambiente do Aluno

O diagrama de casos de uso da Figura 31 foi criado para ilustrar o escopo tratado no ambiente do aluno, no decorrer desta seção serão apresentadas as implementações realizadas e as principais telas associadas estes casos de uso.

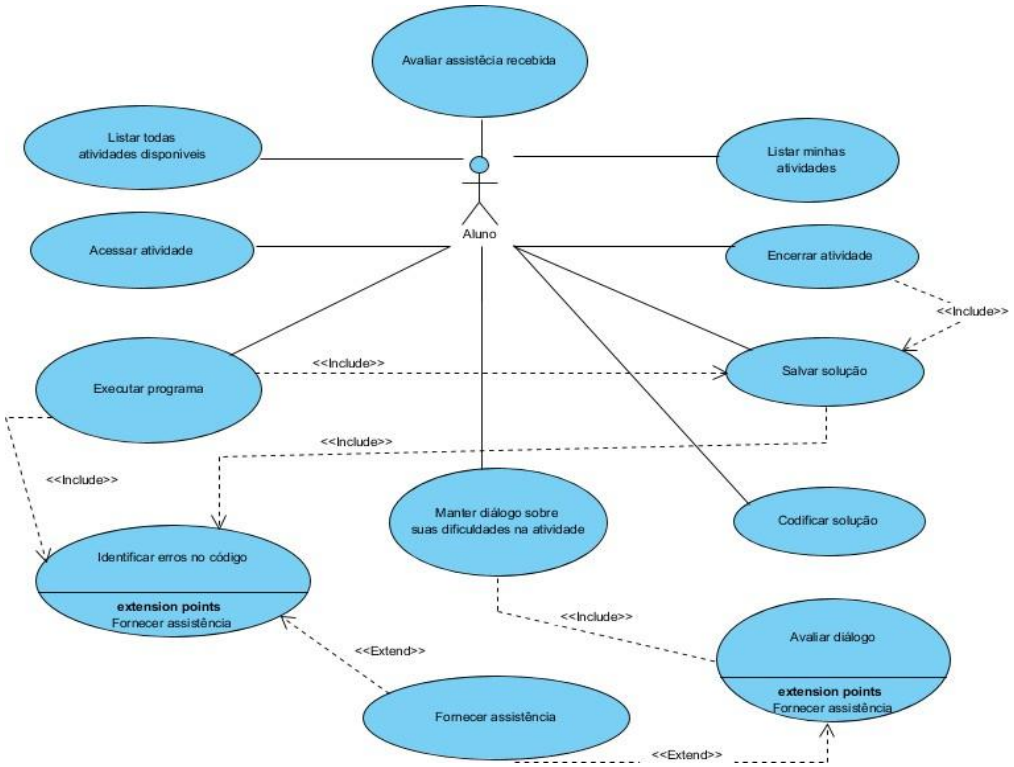


Figura 32 - Casos de usos das funcionalidades implementadas para o aluno.

Fonte: Autoria Própria

7.3.1. O Acesso às Atividades Propostas

Ao entrar no sistema o aluno depara-se com a tela inicial onde são dispostos o menu para o acesso às atividades disponíveis e a área chamada as *"Minhas Atividades".*

A partir do acesso a uma atividade o aluno poderá clicar no botão *"Resolver agora!"*, conforme apresenta a Figura 32. No decorrer do ciclo de vida de uma atividade o botão disposto nesta tela muda para as seguintes ações: *"Continuar de onde parei! "* ou para *"Ver avaliação do professor",* este último caso somente ocorre caso o aluno já tenha concluído a atividade e ela tenha sido avaliada pelo professor.

A partir do acesso a uma atividade o aluno poderá clicar no botão "*Resolver agora!*", conforme apresentado na Figura 32. No decorrer do ciclo de vida de uma atividade o botão disposto nesta tela muda para as seguintes ações: "*Continuar de onde parei!* " ou para "*Ver avaliação do professor*", este último caso somente ocorre caso o aluno já tenha concluído a atividade e ela tenha sido avaliada pelo professor.

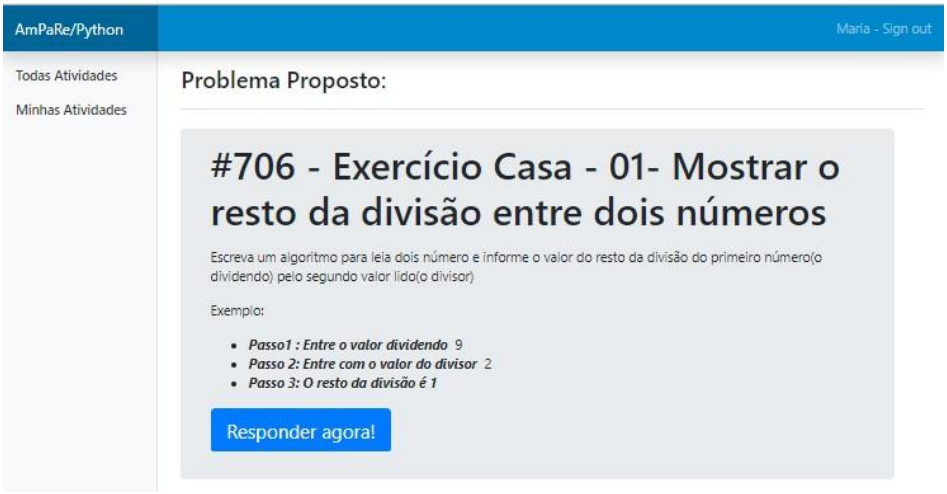


Figura 33 - Tela de visualização da atividade proposta.

Fonte: Autoria Própria

7.3.2. A Área de Resolução, Execução e Conversação

Quando um aluno abre uma atividade para resolvê-la são disponibilizados o ambiente de codificação, a área de conversação e a área de execução do código desenvolvido.

Figura 34 - Tela da área desenvolvimento e de conversação.

Fonte: Autoria Própria

Embora o aluno possa utilizar dos recursos de codificação, de conversação e de execução do código em qualquer ordem, são apresentados os passos de 1 a 4

(Figura 33) com o objetivo de exemplificar um fluxo possível na ordem dos eventos e as funcionalidades envolvidas nestes passos. São eles:

- **Passo 1:** O aluno inicia a atividade, ao perceber uma dificuldade ele solicita uma ajuda escrevendo essa dificuldade na área de conversação. A necessidade do aluno pode ser uma dúvida sobre o enunciado, sobre a sintaxe de um certo comando, sobre a semântica ou comportamento de um dado operador ou função da linguagem e etc. A resposta adequada dependerá sempre da existência de um assistente condizente com a necessidade informada;
- **Passo 2:** Um assistente retorna à solicitação de ajuda e o aluno avalia o *feedback* recebido;
- **Passo 3:** Ao solucionar as dificuldades o aluno volta ao seu código para modificá-lo ou continuar desenvolvendo;
- **Passo 4:** Na aba de controle da atividade, o aluno poderá visualizar o enunciado completo da atividade sem sair da página, salvar a versão atual de seu código ou executá-lo. Também nesta área é disponibilizado o Console Python online, onde o aluno poderá executar e experimentar comandos e ideias sem comprometer o desenvolvimento de sua atividade. O Console Python é disponibilizado na mesma página, sem a necessidade do aluno sair da solução. Ainda no passo 4, caso seja identificado algum erro ao salvar ou executar o programa do aluno, um Assistente de Erros é acionado na área de conversação para dialogar com o aluno sobre a ocorrência identificada.

Embora em menor destaque, as seguintes atividades abaixo também são disponibilizadas no ambiente de codificação apresentado na Figura 33:

- Auto endentação do código;
- Auto complementação de código, por exemplo, o fechamento de parênteses e chaves, nomes de funções nativas do Python e outros recursos locais criados pelo aluno;
- Destaque nas palavras chaves da linguagem.

7.3.3. Área de Conversação e de Avaliação do Suporte Recebido

A área de conversão, conforme apresentado no lado esquerdo da Figura 33 e agora em detalhes na Figura 34, é responsável por oferecer ao aluno as funcionalidades previstas nos casos de uso *“Manter diálogo sobre suas dificuldades na atividade”* e caso de uso *“Avaliar assistência recebida”*. Ao receber um *feedback* de um assistente disponível o aluno pode escolher avaliá-lo com as seguintes opções: *"Sim, me ajudou! "*, *"Entendi, mas ainda preciso de ajuda"* e por último, a *opção "Não me ajudou em nada"*, na ordem da esquerda para direita dos ícones abaixo da conversação apresentados Figura 34.

Uma conversação pode ser iniciada a qualquer momento pelo aluno e também durante a identificação de ocorrências de erros no código do aluno. Caso uma assistência seja avaliada como "*Entendi, mas ainda preciso de ajuda* “ ou "*Não me ajudou em nada”, o sistema enviará a dificuldade para professor complementá-la ou corrigi-la, e* se for o caso escolher atender pessoalmente o aluno.

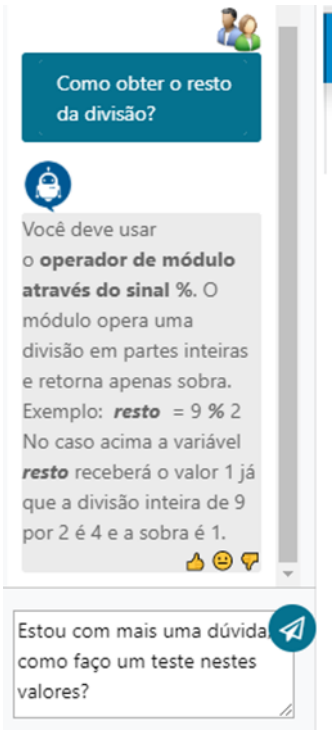


Figura 35 - Tela da área de conversação e de avaliação do *feedback* obtido.

Fonte: Autoria Própria

Conforme ilustrado na Figura 34, buscou-se criar uma área de conversação de fácil utilização cujo a identidade visual se assemelhasse aos principais *chatbots* e aplicativos de comunicação instantânea tais como o *Facebook Messenser*[5] e o *WhatsApp Web*[6].

7.4. A Implementação do Assistente Conversacional

O assistente conversacional foi construído através da classe *AssistenteConversacional*, uma realização da interface *IAssistenteBase* conforme descrito na arquitetura apresentada no Capítulo 5.

O mecanismo de inferência dos assistentes conversacionais foi desenvolvido com base no algoritmo *Naive Bayes* já referenciado no Capítulo 3. Sua implementação visou apoio na identificação das intenções relacionadas a um texto e tomada de decisão do assistente conforme a intenção relacionada.

Conforme ilustra o diagrama da Figura 35 a classe *AssistenteConversacional* utiliza a classe *AnalisadorIntencaoConversacao* que por sua vez identifica as intenções relacionadas um texto através do uso da classe *ClassificadorNaiveBayes*.

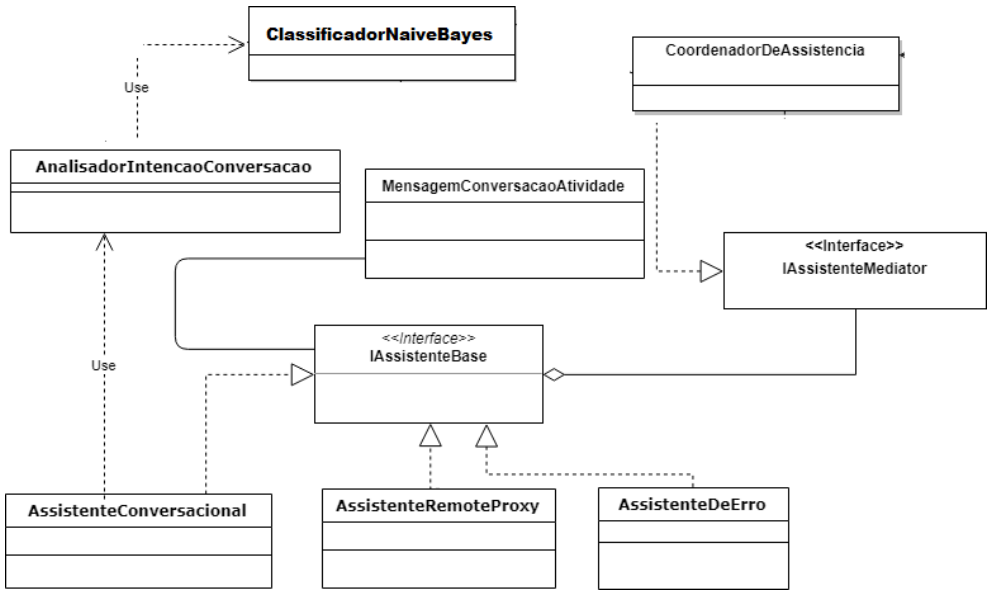


Figura 36 - Diagrama de classe envolvidas no tratamento da conversação.

[5] https://www.facebook.com/messenger/
[6] https://web.whatsapp.com

Fonte: Autoria Própria

Cada assistente tem em sua base de conhecimento um conjunto de mensagens enviadas pelos alunos e também pelo professor ao compartilhar a sua experiência sobre as eventuais dificuldades que poderão ser atendidas por um dado assistente. Além disso, cada assistente possui em sua base interna as ações decorridas que ele desempenhou nos atendimentos anteriores e as dificuldades informadas em cada um desses atendimentos, as avaliações dos alunos e dos professores para essas ações.

O processo de identificação de uma intenção provável contida na mensagem do aluno é realizado seguindo os passos ilustrados na Figura 36 e são explicados em seguida.

A mensagem do aluno indicando a sua dificuldade o ponto inicial para que o assistente realize a identificação da intenção contida nesta mensagem. Em seguida, o assistente utiliza um objeto da classe AnalisadorIntencaoConversacao para a tarefa de classificação de texto.

Ao ser instanciado um objeto da classe *AnalisadorIntencaoConversacao* ele recebe a mensagem do aluno e um conjunto de mensagens já rotuladas, oriundas da base de conhecimento do assistente, conforme ilustra a Figura 36.

Figura 37 - Processo de classificação em intenções prováveis.

Fonte: Autoria Próprio

No processamento do texto é realizada, em primeiro passo, a separação dos termos contidos na frase, a remoção de *Stop Words* e por último o processo de *Stemmer* do texto. Para a realização do *Stop Words* e o do processo de *Stemmer* sobre o texto foram utilizadas as classes *nltk.stem.RSLPStemmer* e *nltk.corpus.stopwords,* ambas oriundas da biblioteca *NLTK* (*Natural Language Toolkit*) disponível para *Python*.

No passo seguinte (Figura 36) é executado o processo chamado *"Recuperação de caso já rotulados"*, nele é recuperado um modelo vetorial que representa a base de conhecimento do assistente e seus históricos de ações para cada mensagem tratada e bem avaliada. Esse modelo vetorial é representado internamente através da classe *CountVectorizer* do pacote *Scikit-Learn* disponível para Python.

Finalmente, no processo de classificação de texto, parte inferior da Figura 36, são fornecidos ao *ClassificadorNaiveBayes* a mensagem do aluno já representada no modelo vetorial com as respectivas frequências e também a base de conhecimento do assistente devidamente representada vetorialmente e rotulada. Ao término do processamento *ClassificadorNaiveBayes* retorna os rótulos e as respectivas probabilidades de cada rótulo ligado a mensagem do aluno.

Para a troca de informações entre os assistentes, visando a cooperação entre eles no atendimento a uma dificuldade informada, foi empregada a analogia do Quadro-Negro, implementado a partir da classe *CoordenadorDeAssistencia* (Figura 20), onde os múltiplos assistentes enquanto agentes inteligentes trabalham isoladamente, entretanto, compartilham de um local comum onde buscam as tarefas e registram suas atividades sobre elas.

Uma instância da classe *CoordenadorDeAssistencia* é criada por aluno no contexto de cada atividade em desenvolvimento e serve para comunicar as novas demandas do aluno a todos assistentes disponíveis na tarefa e intercambiar o diálogo entre os assistentes e o aluno durante a atividade.

7.5. A Implementação do Assistente de Erros

Nesta prova de conceito foi implementado um tipo de assistente analisador de código reapresentado pela classe *AssistenteDeErro*. O assistente de erro é responsável somente pela a análise de erros contidos nos códigos-fonte escrito em Python. O mecanismo de tratamento dos erros foi desenvolvido com base no catálogo de erros fornecido pela biblioteca *PyLint*.

Foram listados e traduzidos 231 eventos de erros e avisos reconhecidos pelo *Pylint* ao avaliar um código-fonte em *Python*. Essa tradução visou unicamente auxiliar o professor na seleção desses eventos durante a especialização de um dado assistente para o erro, sendo facultativo ao professor personalizar as mensagens mais adequadamente ao instanciar um assistente no contexto da atividade onde tal evento é esperado.

O processo de verificação de código do aluno e o tratamento de erro são ilustrados na Figura 37.

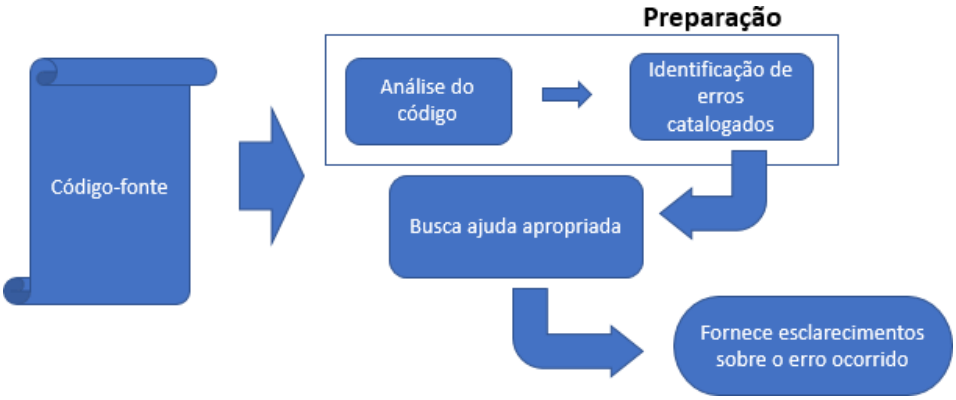


Figura 38 - Processo de classificação em intenções prováveis.

Fonte: Autoria Própria

Logo que o código-fonte é salvo ou há a tentativa de execução sua versão mais atual é encaminhada para análise (Figura 37), neste momento uma instância do *Pylint* é executada e o código-fonte é submetido para análise. Ao tentar rodar o código o *Pylint* analisa-o e retorna uma listagem de todos os erros identificados. No passo seguinte, a listagem de erros é comunicada é avaliada por cada assistente que então busca internamente se ele possui alguma ação prevista. Na ocorrência de assistentes adequados para o tratamento do erro sua seleção é realizada partir da

classe *CoordenadorDeAssistencia* que então coordena a interação desses assistentes na conversação com o aluno. Nesta prova de conceito, cada assistente somente tratava um tipo de erro, portanto foram tratados cenários onde eles competiriam pelo mesmo atendimento.

Os 231 eventos disponíveis para a criação de assistente estão distribuídos nas seguintes categorias:

- 90 eventos que indicam um erro de execução do programa;
- 8 eventos que indicam erros graves (*fatal erros*) no programa;
- 70 eventos que indicam uma violação aos padrões do *Python 3.7* ;
- 72 eventos que indicam um código mau construído, onde o professor poderá criar dicas específicas.

Para cada evento disponível o professor poderá criar estratégias específicas de mediação por atividades, por exemplo, explicar em maiores detalhes como o aluno poderá corrigir um dado caso de erro ou mesmo solicitar que o aluno inicie um diálogo conversacional sobre um dado assunto onde é identificado a necessidade de melhor entendimento.

7.6. Conclusões Finais sobre o Capítulo

Nesta Prova de Conceito cada assistente foi especializado para tratar uma única intenção, sendo assim, cada rótulo provável identificado no *ClassificadorNaiveBayes* representou um uma instância específica de um assistente conversacional, muito embora com pequenas alterações um assistente poderá, em seu mecanismo de inferência, tratar e rotular diversos outros tipos de ações e assim estender seu leque de mediação com o aluno.

A implementação realizada buscou dá as condições para que o professor obtivesse a autonomia na especialização das instâncias dos assistentes disponíveis, onde cada uma dessas instâncias, embora herdassem as características gerais do assistente, poderiam fornecer *feedback* variados e evoluírem a sua base de conhecimento separadamente conforme treinamento realizado pelo professor e pela

experiência adquirida no suporte fornecido nas atividades a qual elas estariam vinculadas.

Foi destacada nesta prova de conceito a capacidade do professor compartilhar a sua experiência com os assistentes, permitindo que os assistentes também adquirissem o estilo e modo de mediação do professor ao longo do tempo.

Foram disponibilizados recursos para preservar a autonomia do professor na seleção dos assistentes por atividades, de forma que eles atuassem somente dentro do escopo definido pelo professor e nas dificuldades que o docente os tenha escolhido.

A utilização do algoritmo *Naive Bayes* permitiu a implementação de um analisador de intenção que foi utilizado no mecanismo de inferência dos assistentes conversacionais no tratamento das mensagens dos alunos ao expressar as suas dificuldades, embora implementações futuras ou de outros assistentes possam utilizar outras abordagens na identificação da dificuldade, inclusive utilizando os dados de contexto disponíveis fachada *ContextoFacade*. O uso do *Pylint* permitiu avaliar o código-fonte do aluno e identificar diversos tipos de erros ou avisos sobre o mal-uso da linguagem de programação. Além disso, foi propiciado os mecanismos para que o professor especialize e parametrize todas as mensagens fornecidas pelos assistentes conversacionais de erros, dando condições para que os assistentes incorporem a forma de mediação do professor ao longo do tempo.

A implementação realizada nesta prova de conceito permitiu validar a proposta deste trabalho sob a perspectiva da arquitetura de softwares e padrões estabelecidos, atendendo aos requisitos estabelecidos no Capítulo 5. As abordagens computacionais empregadas permitiram a implementação do ambiente disponível inteiramente via web, possibilitando, dentre outras características, que os alunos realizem a codificação de seus programas e os execute diretamente no navegador de Internet e sem a necessidade de instalações de componentes.

8. Avaliação do Trabalho

Este capítulo tem como objetivo avaliar o impacto da Prova de Conceito inserido dentro de contexto real na aprendizagem de programação. Portanto, o ambiente foi inserido para uso num curso de intensivo para alunos da disciplina de algoritmos e lógica de programação, onde foi utilizado por 17 alunos num período de quatro semanas. No decorrer deste capítulo serão apresentados os detalhes ligados a este experimento, os resultados obtidos, tanto sobre o ponto de vista qualitativo na perspectiva dos usuários do ambiente, quanto sob os aspectos quantitativos associados à performance obtida no comportamento esperado do sistema.

Primeiramente, na Seção 8.1 será apresentada a metodologia empregada nesta avaliação, o perfil do grupo selecionado para experimento, o perfil do curso, a preparação do ambiente e os métodos utilizados na coleta dos dados analisados.

Em seguida na Seção 8.2 será avaliada a aceitação da ferramenta pelos 17 alunos do curso Técnico em Informática do Centro Estadual de Educação Tecnológica do Estado do Espírito Santo (CEET - Vasco Coutinho) selecionados para o experimento. Ainda nesta Seção serão avaliados os aspectos qualitativos obtidos através dos questionários aplicados semanalmente durante o experimento e serão avaliados os aspectos quantitativos relacionados às assistências fornecidas pelo *AmPaRe* durante a realização das atividades no período.

Em seguida, na Seção 8.3 será feita uma análise quantitativa a fim de averiguar a eficiência do algoritmo *Naive Bayes* quando aplicado na construção dos assistentes conversacionais.

Finalmente, na Seção 8.4 serão realizadas as conclusões finais deste Capítulo.

8.1. Metodologia Aplicada

O experimento foi realizado a partir da promoção de um curso intensivo na disciplina de algoritmos e lógica de programação através da linguagem de programação *Python*, destaca-se que os alunos não utilizavam o *Python* anteriormente, tendo o primeiro contato com essa linguagem a partir do experimento.

O público alvo do curso foram alunos do curso Técnico em Informática do Centro Estadual de Educação Tecnológica do Estado do Espirito Santo (CEET - Vasco Coutinho). Para seleção foi disponibilizado um formulário de inscrição onde o aluno preenchia com seus dados e informavam a motivação para fazerem o curso. Dado as dificuldades de alguns alunos, houveram indicações diretas de seus professores para a realização do curso.

Foram 41 inscritos, dos quais 17 alunos foram selecionados para o curso, os critérios para seleção foram:

- Alunos que se declararam com muita dificuldade em lógica de programação;
- Alunos indicados por seus professores por possuírem muita dificuldade na disciplina de Algoritmos e Lógica de Programação do referido curso técnico;
- Alunos que estavam repetindo a disciplina ou que avançaram para outros módulos do curso, porém com notas abaixo de 7.0 (sendo necessário o mínimo de 6.0 na média para ser aprovado).

Não foram selecionados os alunos que apenas desejavam aprender uma nova linguagem de programação ou aqueles que não se declararam com dificuldades na disciplina regular.

8.1.1. Perfil do Curso Realizado

O curso proposto possuiu carga horária de 40 horas, sendo 20 horas distribuídas em aulas presenciais compostas de teoria e práticas em laboratório. As demais 20 horas foram distribuídas em atividades extraclasse para a realização durante a semana, onde os alunos solucionavam os exercícios a partir de casa através do acesso ao *AmPaRe* pela Internet.

A ementa e a ordem de assuntos introduzidos do curso são uma adaptação dos modelos usados em Pimentel e Omar (2008) e Berssanette e Francisco (2018). O Quadro 4 apresenta a distribuição desses assuntos durante as semanas do curso.

Quadro 4 - Distribuição semanal do conteúdo da ementa.

	Tópico
1***Semana**	Declaração de variáveis
	Atribuição
	Entrada e saída de dados
	Operadores matemáticos
	Conversão de dados
2***Semana**	Operadores Lógicos
	Estrutura condicionais
3***Semana**	Listas com *Python* e conceito de vetores
	Estruturas de repetição com lista (*FOR* do *Python*)
	Estrutura de repetição (*While* do *Python*)
4***Semana**	Funções nativas do *Python*
	Criação de funções e procedimentos

Fonte: Autoria Própria

As atividades desenvolvidas semanalmente seguiram a seguinte ordem:

- Aula teórica sobre um grupo de unidades do curso;
- Codificação de exemplos no ambiente, já utilizando os assistentes;
- Realização de exercícios em laboratórios;
- Disponibilização de exercícios extras para serem desenvolvidos em casa;
- Aplicação de questionário semanal, disponibilizado para o aluno responder ao término da aula ou durante a semana.

Os exercícios propostos foram desenvolvidos em ordem progressiva de complexidade, sendo composto de conhecimentos desenvolvidos durante a aula e o de revisão habilidades desenvolvidas em aulas anteriores. Os níveis de complexidade dos exercícios eram elevados na medida que diversas habilidades deveriam ser combinadas para a resolução de um dado problema.

8.1.2. Métodos de Coleta de Dados

O curso foi iniciado com apresentação da proposta da pesquisa e do experimento. Na ocasião, o ambiente foi apresentado aos alunos, assim como as instruções de uso das funcionalidades de conversação, de avaliação do *feeback* recebido, a área codificação e de submissão da atividade. Essa apresentação também aconteceu em

momentos pontuais a fim de orientar aos alunos como melhor utilizar as funcionalidades disponíveis.

Durante a realização das atividades em laboratório os alunos foram convidados a utilizar o ambiente de conversação sempre que necessitasse de alguma ajuda, embora o professor também estivesse presente. Tanto a utilização do ambiente de conversação quanto a avaliação do *feedback* não foram estimulados, a fim de se obter o comportamento natural e espontâneo do aluno no ambiente.

Foram disponibilizados semanalmente no ambiente um questionário onde os alunos avaliavam o ambiente, a metodologia aplicada e o seu desempenho durante a semana. Foram aplicadas 4 questões objetivas baseadas na escala de Likert[7] , oferecendo aos estudantes a mesma quantidade de respostas negativas e positivas ao responder o critério avaliado, assim como empregado em (ALVES; JAQUES, 2014; SILVA et al., 2016). Duas questões discursivas foram dispostas no questionário para que o aluno pudesse realizar as suas críticas e as suas sugestões sobre o *feedback* recebido através dos assistentes e sobre a metodologia utilizada.

8.1.3. Preparação do ambiente de suporte

Foram criadas 32 instâncias dos assistentes conversacionais para o experimento. Cada instância foi especializada para atendimento a um dado tipo de dúvida prevista com base no programa do curso. As dúvidas variavam desde as mais simples, como por exemplo, saber como criar uma variável, até as questões mais elaboradas, por exemplo, como testar se um valor é par ou ímpar ou como percorrer uma lista de dados. Outros tipos de dificuldades previstas foram:

- Dúvidas sobre tipos de dados em *Python*;
- Dúvidas sobre como iniciar a atividade;
- Dúvidas sobre o enunciado;
- Dúvidas sobre operadores matemáticos;
- Dúvidas sobre o uso de operadores relacionais;

[7] A escala de Likert é um tipo de escala de resposta psicométrica usada habitualmente em questionários, e é a escala mais usada em pesquisas de opinião. Ao responderem a um questionário baseado nesta escala, os perguntados especificam seu nível de concordância com uma afirmação.

- Dúvidas sobre uso do operador de módulo, este operador em específico pela devido o número de dificuldade recorrente durante os exercícios;
- Dúvidas sobre uso de lista e como percorrê-la com a estrutura FOR.

As perguntas iniciais poderiam ser elencadas e registradas pelo próprio docente numa necessidade de carga inicial, no entanto decidimos treinar os assistentes com uma base em 197 formas variadas de perguntas para as classes acima, obtidas através da consulta à fóruns online de dúvidas sobre lógica de programação e sobre o uso da linguagem de programação Python por alunos iniciantes.

Quanto aos assistentes de erros, foram criadas instâncias para os 231 eventos de erros previstos (conforme mencionados na Seção 7.5). Sendo assim, a cada ocorrência de um destes eventos um assistente seria encarregado para prestar o atendimento. Ao preparar cada instância de assistente de erro foram registradas informações em português e com níveis de detalhes mais ajustado ao contexto curso, essas mensagens foram criadas com base na documentação do *Pylint* para cada evento possível de erro.

8.2. Avaliação do Uso do Ambiente para Resolução das Atividades

Nesta seção discutidos os resultados da análise qualitativa obtida através de questionários respondidos semanalmente pelos alunos e a realizada a análise quantitativa dos dados gerados pelo ambiente em relação as assistências fornecidas.

8.2.1. A Avaliação Qualitativa

Primeiramente foi avaliada a opinião do aluno sobre a metodologia empregada em sala de aula através do uso do *AmPaRe* empregado no auxílio ao professor no atendimento das demandas de suporte. A pergunta versava sobre comparação com o método tradicionalmente aplicado na disciplina cursada na instituição. A saber, o curso de algoritmos e lógica de programação do CEET é consistido de aulas teóricas desenvolvidas em sala e da realização de atividades em laboratório. Durante as aulas em laboratório o professor atende as dificuldades ocorridas e quando

necessário retoma algum conceito já abordado em sala aplicando exemplos para toda a turma.

Ao avaliar a aceitação do aluno quanto a mudança da metodologia de ensino através do uso do *AmPaRe* foi possível verificar uma gradativa aceitação do aluno. Conforme o gráfico da Figura 38, na primeira semana 41.2% alunos concordavam plenamente com o uso da metodologia apoiada pelo *AmPaRe*, sendo que 47.7% concordavam parcialmente. Ao término da 4a semana, houve uma convergência para concordância plena em 50% e uma concordância parcial de 50% dos alunos.

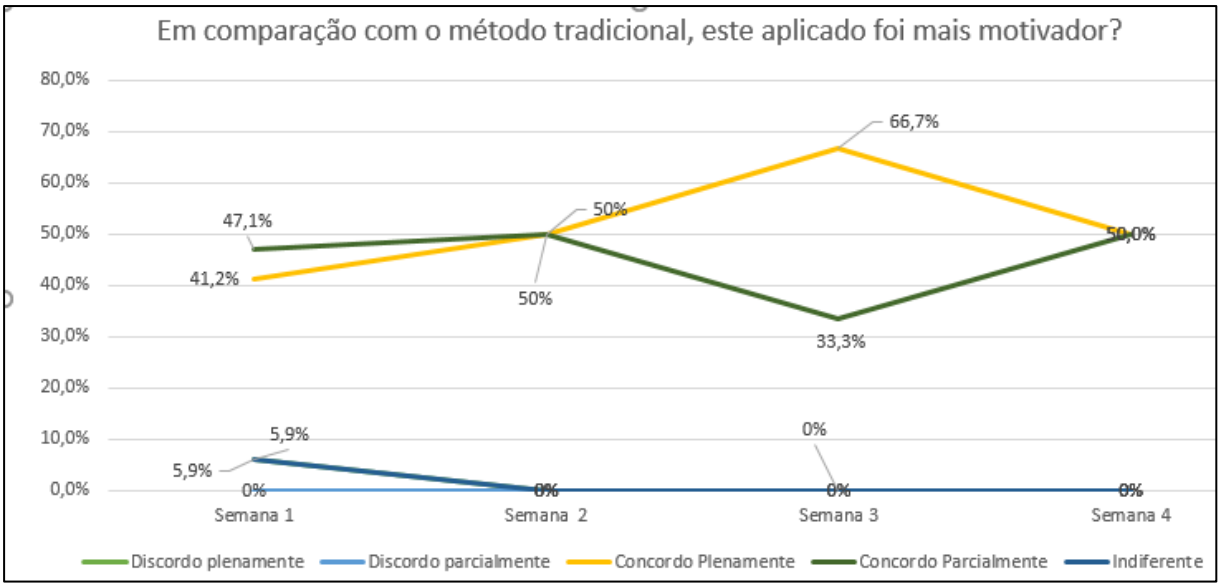


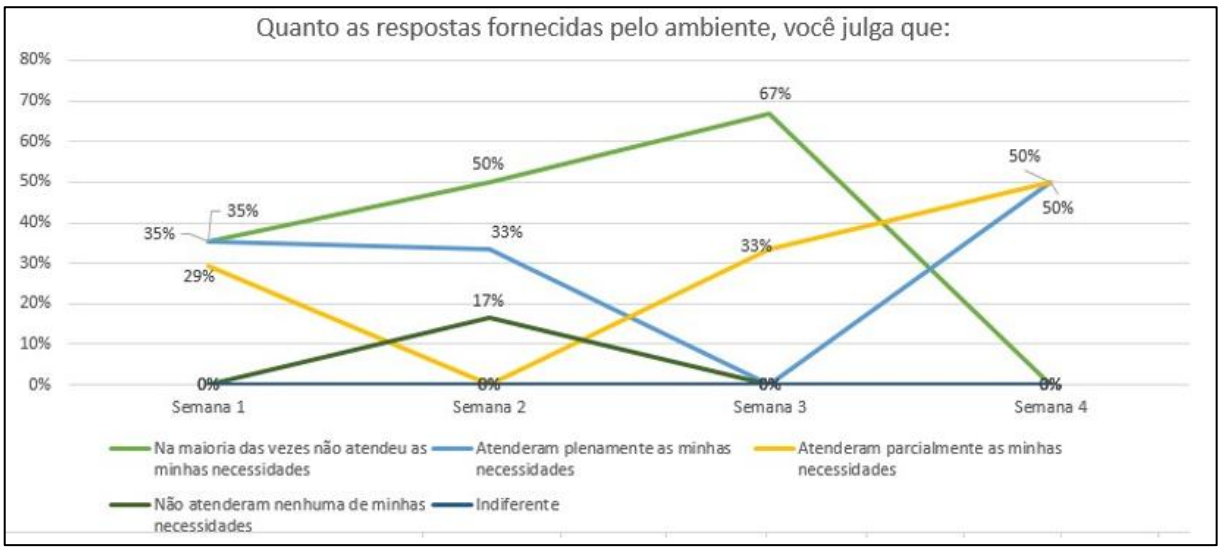
Figura 39 – Gráfico indicativo da opinião do aluno sobre inserção do *AmPaRe* na metodologia de ensino.

Fonte: Autoria Própria

A segunda questão do questionário buscou avaliar qualitativamente se as respostas fornecidas pelos assistentes conversacionais atenderam as necessidades dos alunos. Conforme apresenta o gráfico da Figura 39, observa-se que inicialmente respostas fornecidas pelos assistentes atendiam plenamente as necessidades em 35% dos alunos e parcialmente em 29% deles.

A medida que o curso foi avançando em complexidade o número de alunos insatisfeitos com as respostas aumentaram, passando de 35% na primeira semana

para 67% na terceira semana, posteriormente esse número foi reduzido, sendo que na última semana do experimento o número de alunos plenamente satisfeitos subiu para 50% enquanto que os outros 50% dos alunos afirmavam ter algum nível de



interesse pelos *feedbacks* fornecidos pelo ambiente. O gráfico apresenta na Figura 39 ilustra esses comportamentos.

Figura 40 - Avaliação dos alunos quanto ao suporte fornecido pelo *AmPaRe*.
Fonte: Autoria Própria

Dado o índice de insatisfação dos alunos com os *feedbacks* recebidos entre a segunda e a terceira semana do experimento, foi realizada uma consulta às sugestões e críticas desses alunos no período. Os nomes foram ocultados para preservar o anonimato dos alunos. Seguem os comentários obtidos:

- ***Aluno A:*** *Está em bom caminho. Seria legal que eu digitasse apenas um termo e tivesse todas informações a respeito dele e aplicação prática do mesmo;*
- ***Aluno B:*** *simplificar mais ainda....(entendido como facilitar ainda mais o aprendizado)*
- ***Aluno C:*** *precisa melhorar muito, nas maiorias das vezes não me ajuda e tenho que recorrer ao professor, porém é interessante o modo que nos facilita na hora de resolver o programa quando há resposta coerente a pergunta que fazemos;*

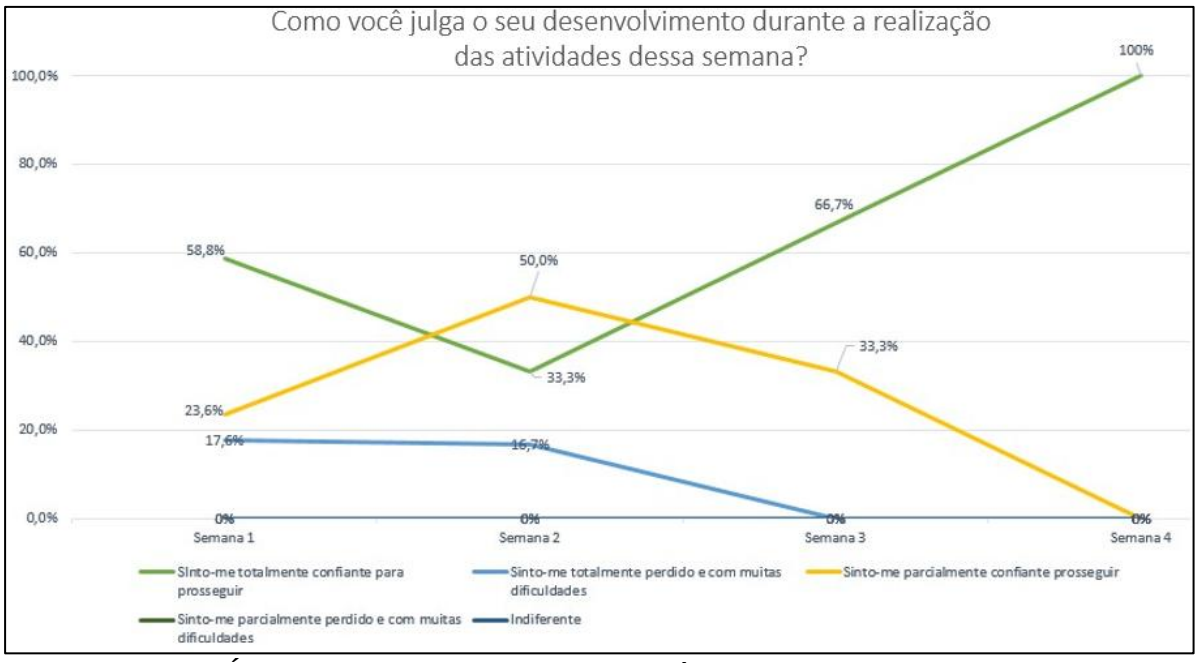
- ***Aluno D:*** *Não é tão favorável.*
- ***Aluno E:*** *Me auxiliou nas dúvidas.*
- ***Aluno F:*** *Precisa colocar respostas mais detalhadas.*

Foi observado por meio dos comentários um grau de exigência elevado dos alunos com relação ao suporte recebido pelos assistentes conversacionais, indicando que eles gostariam de respostas que fossem mais diretas na solução de suas dificuldades ou que o sistema detalhasse o *feedback*, explanando gradativamente o assunto. Todavia, ressalva-se que não necessariamente o assistente tenha respondido errado ao aluno, pois há de se considerar que o *feedback* fornecido por um assistente é derivado de uma atuação anterior do professor ao instanciá-lo para uma atividade e também compartilhar a sua experiência de atendimento com estes assistentes. Portanto, questiona-se que mesmo que o suporte oferecido tenha sido realizado diretamente pelo professor talvez o docente também poderia ter sido mal avaliado por não atender a expectativa particular desse aluno.

Outra questão a ser refletida refere-se à satisfação do aluno com relação a *feedback* recebido, se este não foi ao encontro a sua real necessidade ou daquela dificuldade previamente idealizada pelo professor ao delegar o assistente. Talvez, o aluno idealizava um tipo de suporte e por mais adequado que fosse o suporte oferecido este poderia não atender as suas expectativas internas. Por exemplo, foi identificado durante o experimento um caso onde o aluno estava desejando uma resposta direta que indicasse exatamente o que ele tinha que fazer, como se fosse uma espécie de "macete" ou uma resposta direta ao problema. Esses casos precisam ser melhor investigados a fim de se criar as estratégias de *feedbacks* que orientem a conduta reflexiva do aluno sobre uma ajuda recebida.

O terceiro critério investigado refere-se a autoavaliação do aluno no decorrer das semanas. Ao realizar o seguinte questionamento: *Como você julga seu desenvolvimento durante as atividades nesta aula?* Se observou um desenvolvimento gradual do aluno durante as semanas e um notável aumento na sua autoestima e satisfação com o seu desempenho na resolução das atividades propostas.

Conforme apresentado no gráfico da Figura 40, na primeira semana 58.8% dos alunos se sentiam positivamente confiantes e estavam dispostos a prosseguir



com disciplina. É importante destacar nesta análise que dos 17 alunos selecionados para o experimento, 100% deles se auto avaliaram como um aluno com muita dificuldade na disciplina e noutros casos eles foram indicados por seus professores como alunos em situação de alto risco de desistência ou de reprovação na disciplina regular de programação na qual eles cursavam.

Figura 41 - Autoavaliação semanal do aluno.
Fonte: Autoria Própria

Observou-se que a partir da segunda semana do curso 23,6% dos alunos que estavam parcialmente confiantes passaram a ser 50%, enquanto que a taxa de alunos que afirmavam estar confusos ou perdidos na disciplina reduziu para 16,7%. Todavia, observou-se que dos 58.8% alunos totalmente confiantes apenas 33.3% deles ainda apresentavam a autoestima elevada na segunda semana. A análise realizada levou a considerar que os aspectos ligados a perda da plena autoestima poderiam está ligada ao aumento de complexidade dos exercícios propostos a partir da segunda semana e ao fato de serem abordados os assuntos mais complexos, tais como a estrutura condicional, de repetição e lista de dados. Todavia, somente

um estudo aprofundado sobre este aspecto poderá responder essas oscilações identificadas.

Por último, foi avaliada a qualidade do suporte recebido pelo aluno durante as ocorrências de erros ao tentar salvar ou executar os seus programas desenvolvidos por atividade. A seguinte questão foi levantada ao aluno: *Como você recebeu ajuda quando não entendeu a causa de um erro informado em seu programa?* Esta questão enunciada também foi baseada na escala de Likert, entretanto, o número de resposta foi aumentado para compreender todas as formas de suporte que o aluno poderia ter recebido. Foram dispostas 6 opções de respostas, numa escala que indicava se o aluno foi plenamente atendido pelo sistema e de forma satisfatoriamente, em última caso, se aluno não recebeu nenhum suporte durante a sua necessidade de entender um erro ocorrido em seu programa, nem mesmo o apoio de oriundos de seus colegas.

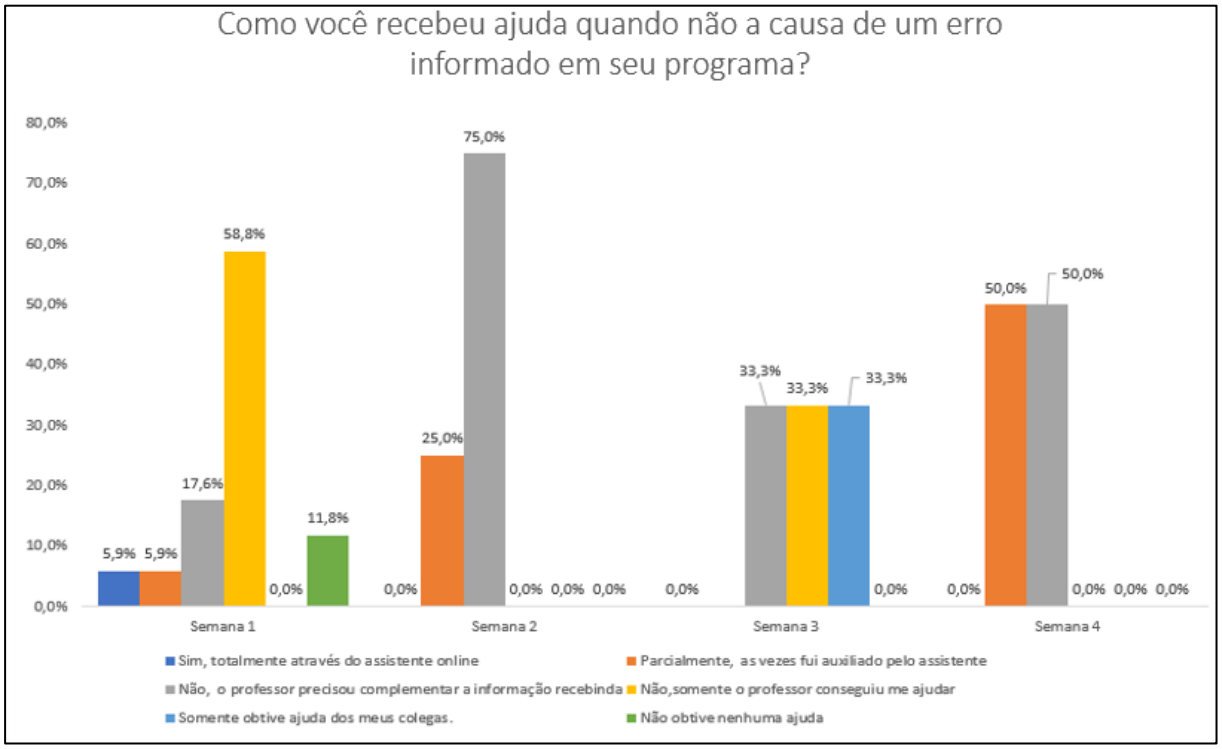


Figura 42 – Gráfico com indicativo das assistências recebidas durante as ocorrências de erros.

Fonte: Autoria Própria

Conforme gráfico da Figura 41, foi observado que durante primeira semana 58,8% dos alunos não assimilaram o suporte fornecido assistentes durante a

ocorrência de um erro em seu programa e apenas 5,9% afirmaram ter sidos auxiliados plenamente pelo sistema durante a ocorrência de erros.

No decorrer do curso foi observada uma diminuição da dependência do professor na explicação das causas de erros. Durante a segunda semana o sistema foi responsável por auxiliar 75% dos alunos, porém sendo necessário o professor complementar o suporte enviado, uma vez que os alunos informaram que o suporte obtido atendera apenas parcialmente às suas necessidades. No decorrer das demais semanas foi observado que professor ainda era necessário na complementação das assistências fornecidas pelo ambiente e um atendimento pleno oferecido pelos assistentes sobre os erros ocorridos não foi alcançado até o final do experimento.

Os resultados apresentados na avaliação qualitativa apontaram a viabilidade inserção e uso do *AmPaRe* no auxílio ao professor e alunos. Embora a necessidade de complementação de respostas, em sua maioria os alunos responderam que o sistema atendia parcialmente ou plenamente as dificuldades informadas e explicavam as causas dos erros ocorridos. Espera-se que no uso contínuo do ambiente a longo prazo que os *feedbacks* fiquem mais pareados com as especificidades da dúvida do aluno e adequados ao contexto de cada atividade. O uso prolongado do ambiente permitirá que outros assistentes sejam instanciados e que o professor ajuste melhor as demandas que serão delegadas aos assistentes no contexto de cada atividade proposta.

8.2.2. Avaliação Quantitativa Das Assistências Fornecidas

Durante o período do curso foram disponibilizados 28 exercícios dos quais 19 foram desenvolvidos em laboratório e 9 foram distribuídos para resolução em casa e durante a semana.

Os alunos recorreram ao ambiente 106 vezes, realizando perguntas ou apontando as suas dificuldades através do ambiente de conversação. Conforme ilustrado no gráfico da Figura 42, houveram em contrapartida, 106 atendimentos, ou seja 100% das demandas tiveram algum tipo de *feedback* para o aluno. Foi constatado, no entanto, que somente 25 dessas assistências recebidas foram avaliadas pelos alunos. Salienta-se que o sistema não exigia que o aluno avaliasse cada interação com os assistentes, esses motivos já foram explicitados na Seção 8.1.2.

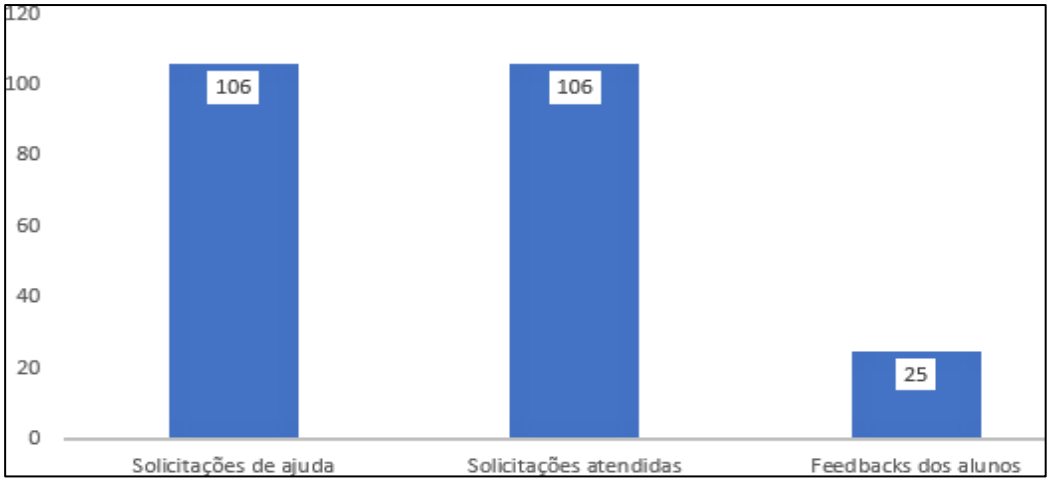


Figura 43 - Número assistências solicitadas, respondidas e avaliadas.
Fonte: Autoria Própria

Foi observado, conforme apresentado na Figura 43, que o *AmPaRe* atendeu positivamente 60% dos atendimentos que foram avaliados pelos alunos. Ao verificar o tratamento realizado pelos assistentes, dos 10 casos mal avaliados, 6 foram atendidos conforme previsto pelo professor, ou seja, a resposta fornecida estava dentro do contexto da atividade e da solicitação realizada pelo aluno, embora o aluno não tenha gostado da ajuda fornecida.

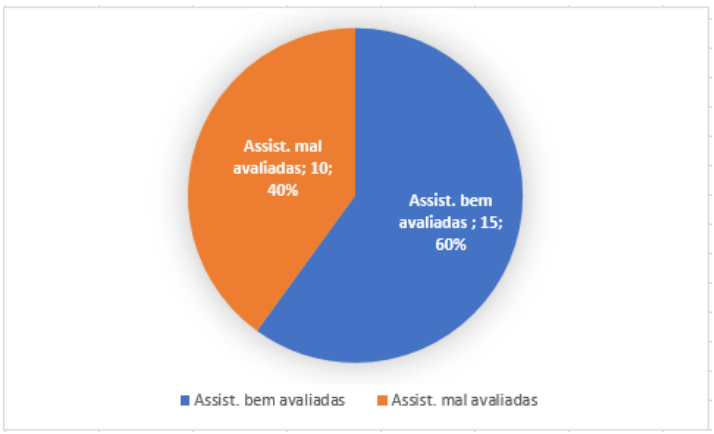


Figura 44 - Análise de desempenho das assistências avaliadas.
Fonte: Autoria Própria

Quando foi avaliado o atendimento realizada pelo assistente e o comportamento esperado pelo professor foi identificado que o *AmPaRe* atendeu adequadamente 21 dos 25 casos avaliados, correspondendo 84% de assertividade.

Quanto a análise quantitativa dos erros ocorridos, foi identificado um total 627 erros ocorridos durante a realização das atividades propostas, onde o sistema atendeu a 100% dessas ocorrências, informando a mensagem indicada pelo professor em complemento ao erro fornecido pelo interpretador *Python*.

Quadro 5 - Categorias de erros ocorridos durante o experimento.

Categoria de Erros	**Total de Ocorrências**
Erro de sintaxe	370
Uso de variáveis inexistente ou com nome errado	115
Conversão inválida	62
Problemas com endentação	33
Erro com indexes das listas de dados	23
Tentativa de somar *String* com tipo numérico	17
Tentativa de acesso de atributo de tipos primitivos	3
Ausência de argumentos obrigatórios em funções	2
Tentativas de usar *index* em variável numérica	1
Tentativa de uso de recurso não implementado no Skupt	1

Fonte: Autoria Própria

8.2.2.1 Considerações Finais sobre a Análise Quantitativa

Nesta seção foi realizada uma análise quantitativa do comportamento do *AmPaRe* durante o experimento realizado com 17 alunos. Das quais 28 atividades foram desenvolvidas, 32 assistentes foram criados para suporte as dificuldades previstas pelo professor. Constatou que o ambiente atendeu a todas as dificuldades registradas pelos alunos no ambiente conversacional, totalizando 106 ocorrências para os assistentes conversacionais e 627 ocorrências para os assistentes de erros. Das 25 assistências avaliadas, constatou-se o desempenho de 60% de assertividade quando esses atendimentos foram avaliados pelos alunos. Quando os atendimentos mal avaliados foram revistos segundo a perspectiva do professor foi

identificado que o ambiente falhou apenas em 4 casos, tendo um desempenho de 84% de assertividade nos *feedbacks* no atendimento às dificuldades informada pelos alunos durante a interação com os assistentes.

8.3. Validação dos Mecanismos de Inferência dos Assistentes Conversacionais

A fim de avaliar quantitativamente se os 32 assistentes criados atenderam as dificuldades as quais estavam delegados e se corresponderam coerentemente ao treino recebido, foram avaliados cada *feedback* oferecido por eles durante o período do experimento. O critério avaliado nesta seção não levou em consideração a satisfação do aluno quanto a resposta recebida, antes foi considerado apenas o comportamento esperado pelo professor ao instanciar o assistente para uma dificuldade esperada.

A análise consistiu da verificação de 106 registros de pedidos de ajuda ocorridos no período do experimento, dos quais em 11 casos, todavia, para estes casos o sistema não tinha condições de atender por se tratar de dificuldades não previstas pelo docente. Conforme ilustra o gráfico apresentado da Figura 50.

Conforme ilustrado na Figura 44, em 57 casos os assistentes atenderam adequadamente, fornecendo o *feedback* condizente com a dificuldade prevista. Esse desempenho correspondeu a uma taxa 54% de acertos.

Os atendimentos equivocados foram aqueles ocorridos quando o assistente forneceu uma ajuda ao invés de outra, foram 38 ocorrências desses casos, correspondendo cerca de 36% na taxa de erros no comportamento dos assistentes.

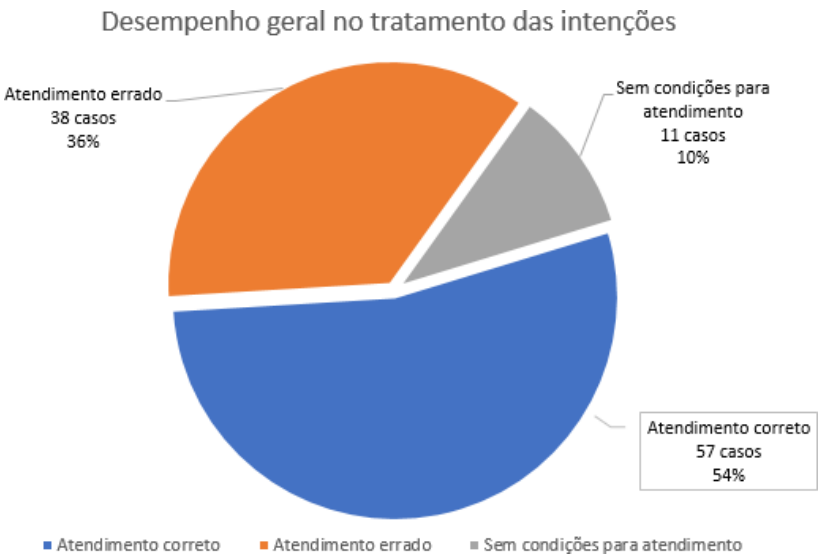


Figura 45 - Desempenho geral no tratamento de intenções.

Fonte: Autoria Própria

Conforme ilustra o gráfico da Figura 45, se fossem apenas considerados os atendimentos realizados dentro dos cenários previstos pelo docente, obtém-se um total de 101 casos de suporte dos quais 60% deles tiveram seus atendimentos bem-sucedidos.

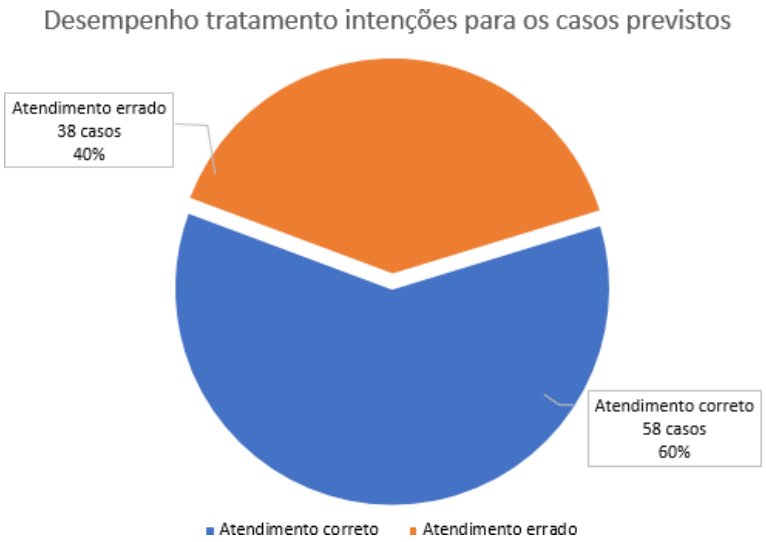


Figura 46 - Desempenho no tratamento de casos previsto.

Fonte: Autoria Própria

Muito embora os dados pesem em favor do atendimento correto, correspondendo um resultado satisfatório em 60% dos casos, há de se considerar

que o período do experimento ainda foi precoce e a quantidade de dados utilizados tanto nos treinos dos assistentes quanto na análise do comportamento das assistentes foram muitos reduzidos para ser ter uma conclusão melhor fundamentada.

Para que os dados apontem um resultado mais sólido faz-se necessário, talvez, o uso do ambiente durante todo um semestre de uma disciplina de introdução a programação ou mesmo o uso por diferentes turmas, no entanto, esta amplitude do experimento não foi possível dentro do tempo dessa pesquisa.

8.4. Considerações Finais do Capítulo

A realização do experimento compreendido num período de 4 semanas permitiu constatar que o ambiente desenvolvido é uma solução promissora e aplicável nas disciplinas de introdução linguagem de programação. O *AmPaRe* foi a única ferramenta utilizada para a codificação das soluções dos exercícios propostos durante o experimento e também foi o único recurso empregado na codificação dos programas desenvolvidos em *Python* durante o período do experimento, sendo disponibilizados neste tempo 32 exercícios e todos foram resolvidos através do ambiente.

Aplicação dos questionários semanais permitiu verificar que o aluno gradativamente fez o uso dos assistentes conversacionais para registrarem as suas dificuldades e obtiveram os *feedbacks* oferecidos pelos assistentes.

Por meio das avaliações realizadas foi possível constatar que sistema desenvolvido é uma solução promissora no apoio aos alunos com dificuldades de aprendizagem nas disciplinas introdutórias de programação. Observou-se um crescimento gradual na satisfação dos alunos com as respostas recebidas no decorrer das semanas do experimento. Enquanto na primeira semana apenas 35% dos alunos informavam estar satisfeitos com os *feedbacks* recebidos, ao término do experimento foi verificado que 100% dos alunos ou estavam plenamente satisfeitos (50%) ou eram parcialmente satisfeitos (50%), sendo que os casos onde o sistema não atendia nenhuma das necessidades dos alunos, sendo que esse índice somente foi relevante até a segunda semana quando atingiu 17%.

Muito embora o professor tenha sido necessário na complementação dos *feedbacks* fornecidos, é compreendido que o ambiente possibilita a especialização de novos assistentes dedicados ao tratamento dos novos casos de suporte não previstos pelo docente. Além disso o ambiente possibilita que o docente reformule as orientações mal avaliadas pelos alunos. Portanto, o uso contínuo do sistema ao longo do tempo permitirá que o professor e assistentes preencham gradativamente as lacunas entre os *feedbacks* fornecidos e as necessidades não atendidas, melhorando assim a performance do ambiente e a diminuição da demanda do docente para os casos recorrentes de atendimento.

9. Conclusão e Considerações Finais

A pesquisa realizada durante esse trabalho permitiu identificar as principais dificuldades de professores e de alunos ocorridas durante um curso de introdução à linguagem de programação. Identificou-se que essas dificuldades são mais evidentes durante a realização das atividades propostas pelo professor e que o docente por muitas vezes fica impossibilitado de atender a todas as demandas desses alunos. Por outro lado, durante a realização dos exercícios propostos os alunos manifestam a suas inquietações e os seus conflitos, por conseguinte, sentem a necessidade de *feedbacks* que permitam que eles avancem no seu aprendizado, todavia, nem sempre o aprendiz é atendido a tempo e seu progresso muitas das vezes fica prejudicado.

O problema de pesquisa consistiu em identificar como prover um ambiente que propiciasse o suporte ao aluno iniciante em linguagem de programação e que otimizasse as condições de atendimento do professor de forma a liberá-lo das demandas recorrentes. Como resultado foram identificados dois tipos de suporte que o professor poderia transferir gradativamente para ambiente proposto: o suporte no atendimento às necessidades relacionadas a análise do código-fonte e os *feedbacks* para as dificuldades informadas pelos alunos através do ambiente de conversação.

O levantamento dos trabalhos correlatos permitiu confirmar a relevância do problema de pesquisa e identificar as características já exploradas em trabalhos anteriores, por fim, permitiu diferenciar a proposta desta pesquisa.

Foi identificado que dentre os trabalhos correlatos nenhum apresentava uma modelagem arquitetural que viabilizasse a expansão do ambiente de programação de forma flexível e com baixo acoplamento. Também foi identificado o pouco envolvimento do professor na evolução e personalização do ambiente, de forma que o docente pudesse enriquecê-lo com a sua experiência ou de acordo com suas novas concepções desenvolvidas no processo de mediação da pergunta e dos erros possíveis uma atividade. Além do mais, as interações nestes ambientes também não forneciam uma conversação com o aluno de forma que o aprendiz pudesse expressar as suas necessidades em linguagem natural, enunciando assim os mais variados tipos de dificuldades que certamente demandariam a atenção do professor.

Esse trabalho resultou numa proposta de arquitetura de software detalhando os aspectos funcionais e arquiteturais com base nos padrões de projetos de software e foi proposto a utilização dos sistemas de agentes inteligentes na constituição dos assistentes.

A arquitetura prevê os mecanismos para a acoplagem de novos assistentes desenvolvidos dentro ou fora da aplicação, flexibilizando a expansão da ferramenta na medida que outros assistentes forem criados para o atendimento aos variado aspectos de cada dificuldade, podendo estes assistentes serem especializados a níveis cada vez mais granulados, por exemplo, um assistente poderá ser dedicado ao tratamento de uma dificuldade específica que ocorre apenas numa certa atividade e evoluir gradativamente toda vez que essa atividade for disponibilizada.

A Prova de Conceito e o experimento realizado permitiram avaliar o objetivo e as hipóteses elencadas no Capítulo 1. Dessa forma, o objetivo geral que norteou esse trabalho foi alçando ao permitir que o professor especializasse os assistentes conversacionais e os assistentes de suporte para a explicação de causas de erros e atendimento das demandas recorrentes de atendimento, permitindo a diminuição da dependência dos alunos pelo professor durante as ocorrências das dificuldades onde já há uma mediação prevista.

O experimento demonstrou a utilização do ambiente durante o curso de intensivo de introdução à linguagem de programação através do uso da linguagem *Python*. Por quatro semanas o ambiente foi exaustivamente utilizado para a realização de 28 exercícios propostos e um total de 17 alunos participaram do experimento.

As avaliações semanais permitiram identificar um gradativo crescimento no uso dos assistentes pelos alunos no decorrer do curso e o ambiente proposto ofereceu a autonomia e a flexibilidade para que o professor definisse qual o tipo de atendimento seria delegado aos assistentes. O ambiente também forneceu as condições necessárias para que o docente pudesse selecionar quais assistentes ficariam responsáveis por acompanhar uma atividade proposta.

Foi confirmada a hipótese que o uso de suporte computacional quando aplicado no fornecimento de *feedback* automático mostra-se como uma solução de

apoio ao professor no atendimento das demandas recorrentes ao tempo que favorece o aprendizado do aluno ao propiciar um suporte no momento de suas dificuldades.

As avaliações qualitativas apontam que 50% dos alunos estavam plenamente satisfeitos com o ambiente, enquanto que os demais 50% apresentavam uma satisfação parcial com o suporte, indicando, segundo a escala de Likert, uma tendência à satisfação pela maioria dos alunos pelo ambiente em uso.

A análise quantitativa do comportamento geral dos assistentes demonstrou que eles agiram corretamente em 60% dos casos onde a dificuldade do aluno já era esperada pelo professor ao configurar os assistentes para uma atividade. Embora sejam dados obtidos de um período curto de uso, eles já apontam as condições necessárias para liberar o professor de muitos casos corriqueiros de pedido de atendimento, ao tempo que dá as condições para que o aluno obtenha o *feedback* no momento de sua dificuldade, mesmo quando o professor não esteja disponível.

A segunda hipótese versava sobre a interação homem-máquina através de um ambiente conversacional, pois viabilizaria ao aluno expressar as suas dificuldades e receber o suporte no tempo adequado. Foi percebido, tanto por meio validação quantitativa quanto pela validação qualitativa, que os alunos se envolveram rapidamente com o ambiente e registravam naturalmente as suas dúvidas na conversação com os assistentes disponíveis na atividade.

Muito embora as críticas recebidas nas validações qualitativas entre a 2ª e 3ª semana do experimento, entendeu-se que o ambiente proporcionou a interação prevista na hipótese visto que em todas as avaliações qualitativas os alunos concordavam parcialmente ou concordavam plenamente com a inserção do ambiente na metodologia de ensino, correspondendo numa taxa média de 52,22% de plena concordância pela inserção do *AmPaRe* durante as atividades.

Ainda que os resultados das análises qualitativa e quantitativa tenham apresentado os dados favoráveis ao uso ambiente, entende-se que a aplicação do ambiente num curso de maior duração ou quando aplicado em mais turmas permitirá aferir com maior precisão o interesse do aluno, bem como o impacto dessas interações no desenvolvimento do aluno ao longo do curso e por fim, aferir com

maior precisão a assertividade dos assistentes no atendimento das dificuldades ocorridas.

A implementação dos assistentes conversacionais e dos assistentes de erros permitiu validar a hipótese levantada sobre a utilização sistemas agentes inteligentes integrados num ambiente de codificação, visto que neste trabalho eles foram responsáveis por monitorarem o desenvolvimento das atividades propostas e também por fornecerem o suporte ao aluno durante dificuldades ocorridas e informadas durante a conversação via texto.

9.1. Trabalhos Futuros

Abaixo são elencados os trabalhos futuros visando a evolução dessa pesquisa e as melhorias no ambiente apresentado. São eles:

- Um ambiente de depuração poderá ser implementado e juntamente com este recurso um novo tipo de assistente poderá ser proposto a fim de acompanhar e oferecer suporte no entendimento do comportamento do programa ao longo da depuração;
- Novos assistentes poderão ser criados com a habilidade de interpretar e tratar as emoções obtidas na conversação com o aluno;
- Os assistentes poderiam empregar técnicas para incorporação de diálogos longos, onde de forma mais proativa, poderiam realizar questionamentos ao aluno ou propor o aprofundamento da conversa num dado assunto;
- Implementar assistentes com a capacidade de lidar com diversos tipos de objetos de aprendizagem (OA) ou mesmo possuir a compatibilidade para integrar-se aos objetos de aprendizagem externos;
- Redes Bayesianas (RB) poderão ser integradas para que o professor e os assistentes possam construir hipóteses sobre as dificuldades dos alunos num certo contexto e registrarem as evidências dessas dificuldades, assim como proposto em Santos, Menezes e Cury (2018). Outra possibilidade é inferir a intenção de uma pergunta com base num modelo do aluno construído a partir de uma RB;

- Outra característica a ser investigada está ligada a implementação e avaliação dos impactos na disponibilização de interações entre alunos, por exemplo, proporcionar que alunos façam indicações de ajudas previamente registradas aos seus pares;
- Trabalhos futuros poderão monitorar o desempenho obtido pelo aluno a partir de cada *feedback* recebido e usar esses dados como reforço nos mecanismos de inferência dos agentes inteligentes empregados na construção dos assistentes;
- Em trabalhos futuros os assistentes não escolhidos para a atividade poderiam monitorar as dificuldades dessa atividade em segundo plano e comunicar ao professor as suas condições de atendimento oferecendo as condições para que o professor reveja a relação de assistentes dedicados a uma atividade;
- O ambiente poderá ser melhorado ao se aplicar mecanismos para agrupar os alunos com as mesmas dificuldades e desenvolver uma interação de suporte em grupo, permitindo a troca de informação entre eles a medida que se desenvolvem.

Referências

AGGARWAL, S. **Flask framework CookBook**. [S.l.]: Packt Publishing Ltd, 2014.

ALVES, F. P.; JAQUES, P. **Um ambiente virtual com *feedba*ck personalizado para apoio a disciplinas de programação**. In: Anais dos Workshops do Congresso Brasileiro de Informática na Educação. [S.l.: s.n.], 2014. v. 3, n. 1, p. 51.

ANDERSON, J. R.; BOYLE, C. F.; REISER, B. J. **Intelligent tutoring systems**. Science, American Association for the Advancement of Science, v. 228, n. 4698, p. 456–462, 1985.

ANDRADE, P. H. M. A. d. **Aplicação de técnicas de mineração de textos para classificação de documentos: um estudo da automatização da triagem de denúncias na CGU**. 2015.

AURELIANO, V. C. O.; TEDESCO, P. d. A.; GIRAFFA, L. M. M. **Desafios e oportunidades aos processos de ensino e de aprendizagem de programação para iniciantes**. In: Congresso da Sociedade Brasileira de Computação. [S.l.: s.n.], 2016. v. 24, p. 2066–2075.

BAEZA-YATES, R.; RIBEIRO-NETO, B. **Recuperação de Informação-: Conceitos e Tecnologia das Máquinas de Busca**. [S.l.]: Bookman Editora, 2013.

BARBOSA, L. S.; FERNANDES, T. C.; CAMPOS, A. M. **Takkou: uma ferramenta proposta ao ensino de algoritmos**. In: XVIII Workshop sobre Educação em Computação (WEI 2011). [S.l.: s.n.], 2011.

BERSSANETTE, J. H.; FRENCISCO, A. C. **Proposta de abordagem prática para o ensino de programação baseada em Ausube**l. In: Brazilian Symposium on Computers in Education (Simpósio Brasileiro de Informática na Educação-SBIE). [S.l.: s.n.], 2018. v. 29, n. 1, p. 398.

BIRD, S.; KLEIN, E.; LOPER, E. **Natural language processing with Python: analyzing text with the natural language toolkit**. [S.l.]: "O'Reilly Media, Inc.", 2009.

BOOCH, G.; RUMBAUGH, J.; JACOBSON, I. **UML: guia do usuário**. [S.l.]: Elsevier Brasil, 2012.

BORGES, K. S.; FILHO, H. B. d. R. **A importância dos grupos de estudos na formação acadêmica**. In: CONGRESSO DA SOCIEDADE BRASILEIRA DE COMPUTAÇÃO. [S.l.: s.n.], 2005. v. 25, p. 2338–2344.

BOULAY, B. D. **Some difficulties of learning to program**. Journal of Educational Computing Research, SAGE Publications Sage CA: Los Angeles, CA, v. 2, n. 1, p. 57–73, 1986.

BUITINCK, L. et al. **Api design for machine learning software: experiences from the scikit-learn project**. arXiv preprint arXiv:1309.0238, 2013.

CAMARGO, A. N. B. de et al. **A pergunta na sala de aula: concepções e ações de professores de ciências e matemática**. 2011.

CARTER, J.; DEWAN, P.; PICHILIANI, M. **Towards incremental separation of surmountable and insurmountable programming difficulties**. In: ACM. Proceedings of the 46th ACM Technical Symposium on Computer Science Education. [S.l.], 2015. p. 241–246.

CARVALHO, M. J. S.; NEVADO, R. A. de; MENEZES, C. S. de. **Arquiteturas pedagógicas para educação à distância: concepções e suporte telemático**. In: Brazilian Symposium on Computers in Education (Simpósio Brasileiro de Informática na Educação-SBIE). [S.l.: s.n.], 2005. v. 1, n. 1, p. 351–360.

CASTRO, T. H. C. de; JÚNIOR, A. N. de C.; MENEZES, C. S. de. **Aprende – um ambiente cooperativo de apoio à aprendizagem de programação**. In: Brazilian Symposium on Computers in Education (Simpósio Brasileiro de Informática na Educação-SBIE). [S.l.: s.n.], 2004. v. 1, n. 1, p. 71–79.

CATAE, F. S. **Classificação automática de texto por meio de similaridade de palavras: um algoritmo mais eficiente**. Tese (Doutorado) — Universidade de São Paulo, 2012.

CHAVES, J. O. et al. **Mojo: uma ferramenta para auxiliar o professor em disciplinas de programação**. In: Congresso Brasileiro de Ensino Superior a Distância, Belém, PA. [S.l.: s.n.], 2013.

COELHO, A. R. **Stemming para a língua portuguesa: estudo, análise e melhoria do algoritmo RSLP**. 2007.

COPPIN, B. **Artificial intelligence illuminated**. [S.l.]: Jones & Bartlett Learning, 2004.

CORONADO, M. et al. **A cognitive assistant for learning java featuring social dialogue**. International Journal of Human-Computer Studies, Elsevier, v. 117, p. 55–67, 2018.

DELGADO, C. et al. **Identificando competências associadas ao aprendizado de leitura e construção de algoritmos**. In: XIII Workshop sobre Educação em Computação. [S.l.: s.n.], 2005.

FALCKEMBACH, G. A. M.; ARAUJO, F. V. de. **Aprendizagem de algoritmos: dificuldades na resolução de problemas**. Anais Sulcomp, v. 2, 2013.

FEKETE, A.; GREENING, T.; KINGSTON, J. **Conveying technical content in a curriculum using problem based learning**. In: ACM. Proceedings of the 3rd Australasian conference on Computer science education. [S.l.], 1998. p. 198–202.

FERNEDA, E. **Recuperação de Informação: Análise sobre a contribuição da Ciência da Computação para a Ciência da Informação**. Tese (Doutorado) — Universidade de São Paulo, 2003.

FERNEDA, E. **Ontologia como recurso de padronização terminológica em um Sistema de Recuperação de Informação**. [S.l.]: 2013a, 2013.

FRANÇA, A. B.; SOARES, J. M. **Sistema de apoio a atividades de laboratório de programação via moodle com suporte ao balanceamento de carga**. In: Brazilian Symposium on Computers in Education (Simpósio Brasileiro de Informática na Educação-SBIE). [S.l.: s.n.], 2011. v. 1, n. 1.

FRANK, E.; BOUCKAERT, R. R. **Naive bayes for text classification with unbalanced classes**. In: SPRINGER. European Conference on Principles of Data Mining and Knowledge Discovery. [S.l.], 2006. p. 503–510.

FREIRE, P. **Pedagogia do oprimido**. 17a. Ed. Rio de Janeiro: Paz e Terra, v. 3, p. 36, 1987.

FREIRE, P. **Educação "bancária" e educação libertadora**. Introdução à psicologia escolar,Casa do Psicólogo São Paulo, v. 3, p. 61–78, 1997.

GAMMA, E. **Padrões de projetos: soluções reutilizáveis**. [S.l.]: Bookman editora, 2011.

GANDA, D. R.; BORUCHOVITCH, E**. A autorregulação da aprendizagem: principais conceitos e modelos teóricos**. Psicologia da Educação. Programa de Estudos Pós-Graduados em Educação: Psicologia da Educação. ISSN 2175-3520, n. 46, 2018.

GERDES, A. et al. **Ask-Elle: an adaptable programming tutor for Haskell giving automated feedback**. International Journal of Artificial Intelligence in Education, Springer, v. 27, n. 1, p. 65–100, 2017.

GIL, A. C. **Como elaborar projetos de pesquisa**. 12. reimpr. São Paulo: Atlas, v. 6, n. 1-1, 2009.

GIRAFFA, L.; MARCZAK, S.; ALMEIDA, G. **O ensino de algoritmos e programação mediado por um ambiente na web**. In: Congresso Nacional da Sociedade Brasileira de Computação (SBC'2003). Campinas, SP, Brasil. [S.l.: s.n.], 2003.

GOMES, A.; HENRIQUES, J.; MENDES, A. **Uma proposta para ajudar alunos com dificuldades na aprendizagem inicial de programação de computadores**. Educação, Formação & Tecnologias-ISSN 1646-933X, v. 1, n. 1, p. 93–103, 2008.

GOMES, C. C. C. et al. **Uma proposta para auxiliar alunos e professores no ensino de programação: O ambiente AIIP**. 2011.

GOMES, M. et al. **Um estudo sobre erros em programação-reconhecendo as dificuldades de programadores iniciantes**. In: Anais dos Workshops do Congresso Brasileiro de Informática na Educação. [S.l.: s.n.], 2015. v. 4, n. 1, p. 1398.

GONZALEZ, M.; LIMA, V. L. **Recuperação de informação e processamento da linguagem natural**. In: XXIII Congresso da Sociedade Brasileira de Computação. [S.l.: s.n.], 2003.v. 3, p. 347–395.

GOTTSCHALK, K. et al. **Introduction to web services architecture**. IBM systems Journal, IBM, v. 41, n. 2, p. 170–177, 2002.

GRANDELL, L. et al. **Why complicate things? introducing programming in high school using python**. In: AUSTRALIAN COMPUTER SOCIETY, INC. Proceedings of the 8th Australasian Conference on Computing Education-Volume 52. [S.l.], 2006. p. 71–80.

GUINARD, D. et al. A resource oriented architecture for the web of things. In: IoT. [S.l.: s.n.], 2010. p. 1–8.

HATTIE, J.; TIMPERLEY, H. **The power of feedback**. Review of educational research, 2007.

HOLANDA, W.; COUTINHO, J.; FONTES, L. **Uma intervenção metodológica para auxiliar a aprendizagem de programação introdutória: um estudo experimental**. In: Anais dos Workshops do Congresso Brasileiro de Informática na Educação. [S.l.: s.n.], 2018. v. 7, n. 1, p. 699.

IEPSEN, E. F**. Ensino de algoritmos: detecção do estado afetivo de frustração para apoio ao processo de aprendizagem**. 2013.

JADE. **JADE - JAVA Agent Development Framework**. 2019. Disponível em: <https://jade.tilab.com/>.

JENKINS, T. **The first language-a case for python?** [S.l.]: Taylor & Francis, 2004.

JUNIOR, G. B. d. S. et al**. Padrões arquiteturais para o desenvolvimento de aplicações multiagente**. Universidade Federal do Maranhão, 2003.

JÚNIOR, G. S.; FECHINE, J. M.; COSTA, E. d. B. **Analogus: Um ambiente para auxílio ao ensino de programação orientado pelo raciocínio por analogia**. XVII WEI, v. 28, 2009.

JUNIOR, S. M. da S.; FRANÇA, S. V. A. **Programação para todos: Análise comparativa de ferramentas utilizadas no ensino de programação**. In: SBC. 25o Workshop sobre Educação em Computação (WEI 2017). [S.l.], 2017. v. 25, n. 1/2017.

KEUNING, H.; JEURING, J.; HEEREN, B. **A systematic literature review of automated feedback generation for programming exercises**. ACM Transactions on Computing Education (TOCE), ACM, v. 19, n. 1, p. 3, 2018.

KOLIVER, C.; DORNELES, R. V.; CASA, M. E. **Das (muitas) dúvidas e (poucas) certezas do ensino de algoritmos**. In: XII Workshop de Educação em Computação. [S.l.: s.n.], 2004.

KÖLLING, M. et al. **The BlueJ system and its pedagogy**. Computer Science Education, Taylor & Francis, v. 13, n. 4, p. 249–268, 2003.

KUTZKE, A. R.; DIRENE, A. **Mediação do erro no ensino de programação de computadores: fundamentos e aplicação da ferramenta Farma-Alg**. In: Anais dos workshops do congresso brasileiro de informática na educação. [S.l.: s.n.], 2016. v. 5, n. 1, p. 1050.

LEMOS, M. F. de et al. **Aplicabilidade da arquitetura MVC em uma aplicação web (WebApps)**. RE3C-Revista Eletrônica Científica de Ciência da Computação, v. 8, n. 1, 2013.

LOKHANDE, P. et al. **Efficient way of web development using python and Flask**. International Journal of Advanced Research in Computer Science, 2015.

LOPER, E.; BIRD, S. **NLTK: The natural language toolkit**. arXiv preprint cs/0205028, 2002.

LUCCA, G. et al. **Uma implementação do algoritmo Naive Bayes para classificação de texto**. Centro de Ciências Computacionais-Universidade Federal do Rio Grande (FURG), 2013.

MANNING, C. D.; RAGHAVAN, P.; SCHÜTZE, H. **Text classification and naive bayes. Introduction to information retrieval**, Cambridge university press, v. 1, n. 6, 2008.

MCILRAITH, S. A.; SON, T. C.; ZENG, H. **Semantic web services**. IEEE intelligent systems, IEEE, v. 16, n. 2, p. 46–53, 2001.

MOREIRA, G. L. 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.

MOREIRA, M. A. **Aprendizagem significativa: da visão clássica à visão crítica (meaningful learning: from the classical to the critical view)**. In: Conferência de encerramento do V Encontro Internacional sobre Aprendizagem Significativa, Madrid, Espanha, setembro de. [S.l.: s.n.], 2006.

MOTA, M. P. et al. **Ambiente integrado a plataforma moodle para apoio ao desenvolvimento das habilidades iniciais de programação**. In: Brazilian Symposium on Computers in Education (Simpósio Brasileiro de Informática na Educação-SBIE). [S.l.: s.n.], 2009. v. 1, n. 1.

MYERS, J.; COPELAND, R. **Essential SQLAlchemy: Mapping Python to Databases**. [S.l.]: "O'Reilly Media, Inc.", 2015.

NAKAGAWA, E. Y. **Uma contribuição ao projeto arquitetural de ambientes de engenharia de software**. Tese (Doutorado) — Universidade de São Paulo, 2006.

NARCISS, S. **Designing and evaluating tutoring feedback strategies for digital learning**. Digital Education Review, Digital Education Observatory (OED), n. 23, p. 7–26, 2013.

NASCIMENTO, P. B. d. et al. **Recomendação de ação pedagógica no ensino de introdução à programação por meio de raciocínio baseado em casos**. Universidade Federal do Amazonas, 2018.

NEVADO, R. A. de; MENEZES, C. S. de; JÚNIOR, R. R. V. **Debate de teses–uma arquitetura pedagógica**. In: Brazilian Symposium on Computers in Education (Simpósio Brasileiro de Informática na Educação-SBIE). [S.l.: s.n.], 2011. v. 1, n. 1.

NOBRE, I. A. M.; MENEZES, C. S. de. **Suporte à cooperação em um ambiente de aprendizagem para programação (Samba)**. In: Brazilian Symposium on Computers in Education (Simpósio Brasileiro de Informática na Educação-SBIE). [S.l.: s.n.], 2002. v. 1,n. 1, p. 337–347.

ODELL, J. **Agent technology: What is it and why do we care?** Enterprise Architecture, Executive report, Cutter Consortium, v. 10, n. 3, p. 1–25, 2007.

OGURI, P.; LUIZ, R.; RENTERIA, R. **Aprendizado de máquina para o problema de sentiment classification**. PUC-Rio. Rio de Janeiro, 2006.

ORENGO, V. M.; HUYCK, C. **A stemming algorithm for the portuguese language**. In: IEEE. Proceedings Eighth Symposium on String Processing and Information Retrieval. [S.l.], 2001. p. 186–193.

PADE. PADE Python Agent Development framework. 2019. Disponível em:<https://pade.readthedocs.io/en/latest/>.

PANCERI, S.; MENEZES, C. de. **Apoio a mediação pedagógica em um debate de teses utilizando técnicas de processamento de texto**. In: Brazilian Symposium on Computers in Education (Simpósio Brasileiro de Informática na Educação-SBIE). [S.l.: s.n.], 2015. v. 26, n. 1, p. 977.

PASCUTTI, M. C. D. **Uma proposta de arquitetura de um ambiente de desenvolvimento de software distribuído baseada em agentes**. 2002.

PEARS, A. et al. **A survey of literature on the teaching of introductory programming**. In: ACM. ACM sigcse bulletin. [S.l.], 2007. v. 39, n. 4, p. 204–223.

PEDREGOSA, F. et al. **Scikit-learn: Machine learning in python**. Journal of machine learning research, v. 12, n. Oct, p. 2825–2830, 2011.

PIEKARSKI, A. E. et al. **A metodologia das maratonas de programação em um projeto de extensão: um relato de experiência**. In: Anais dos Workshops do Congresso Brasileiro de Informática na Educação. [S.l.: s.n.], 2015. v. 4, n. 1, p. 1246.

PIMENTEL, E.; OMAR, N. ***Ensino de algoritmos baseado na aprendizagem significativa utilizando o ambiente de avaliação NetEdu***. SBC, p. 79, 2008.

PIMENTEL, E. P.; OMAR, N.; FRANÇA, V. F. ***Um modelo para incorporação de automonitoramento da aprendizagem em STI***. Brazilian Journal of Computers in Education, v. 13, n. 1, 2005.

PIMENTEL, E. P. et al. **Um modelo para avaliação e acompanhamento contínuo da aprendizagem**. In: Brazilian Symposium on Computers in Education (Simpósio Brasileiro de Informática na Educação-SBIE). [S.l.: s.n.], 2004. v. 1, n. 1, p. 129–138.

PRESSMAN, R. S. **Engenharia de Software-7**. [S.l.]: Amgh Editora, 2011.

PUURULA, A. **Combining modifications to multinomial naive bayes for text classification**. In: SPRINGER. Asia Information Retrieval Symposium. [S.l.], 2012. p. 114–125.

PYLINT. **PYLINT Python Lint Tools**. 2018. Disponível em: <https://pypi.org/project/pylint/>.

PYTHON. **Pytnon The official home of the Python Programming Language**. 2018. Disponível em: <https://www.python.org/doc/>.

QUEIRÓS, R.; LEAL, J. P. Ensemble: **An innovative approach to practice computer programming**. In: Innovative Teaching Strategies and New Learning Paradigms in Computer Programming. [S.l.]: IGI Global, 2015. p. 173–201.

RAABE, A. et al. **Avaliação do feedback gerado por um corretor automático de algoritmos**. In: Brazilian Symposium on Computers in Education (Simpósio Brasileiro de Informática na Educação-SBIE). [S.l.: s.n.], 2015. v. 26, n. 1, p. 358.

RAABE, A. L. A.; SILVA, J. d. **Um ambiente para atendimento as dificuldades de aprendizagem de algoritmos**. In: XIII Workshop de Educação em Computação (WEI'2005). São Leopoldo, RS, Brasil. [S.l.: s.n.], 2005.

RAABE, A. L. A.; SILVA, J. M. C. da. **Um ambiente para atendimento as dificuldades de aprendizagem de algoritmos**. 2005.

RAPKIEWICZ, C. E. et al. **Estratégias pedagógicas no ensino de algoritmos e programação associadas ao uso de jogos educacionais.** RENOTE: revista novas tecnologias na educação [recurso eletrônico]. Porto Alegre, RS, 2007.

RATO, J. P. C. **Conversação homem-máquina. Caracterização e avaliação do estado actual das soluções de speech recognition, speech synthesis e sistemas de conversação homem-máquina**. Tese (Doutorado), 2016.

RESNICK, M. et al. **Scratch: Programming for all**. Commun. Acm, v. 52, n. 11, p. 60–67, 2009.

RODRIGUES, J. P. **Sistemas inteligentes híbridos para classificação de texto**. Dissertação (Mestrado) — Universidade Federal de Pernambuco, 2009.

RODRIGUEZ, A. **Restful web services: The basics.** IBM developerWorks, v. 33, p. 18, 2008.

RUSSEL, S.; NORVIG, P. **Inteligência artificial**. 3a edição. Editora Campus, 2013.

SANTOS, R.; MENEZES, C. de; CURY, D. **Uma arquitetura de tutor inteligente que provê suporte ao diálogo com o aluno iniciante em linguagem de programação**. In: Anais dos Workshops do Congresso Brasileiro de Informática na Educação. [S.l.: s.n.], 2018. v. 7, n. 1, p. 768.

SANTOS, V. d. **Ágora digital: o cuidado de si no caminho do diálogo entre tutor e aluno em um ambiente virtual de aprendizagem**. 2012.

SHIMODAIRA, H. **Text classification using naive bayes**. Learning and Data Note, v. 7, p. 1–9, 2014.

SIGNORETTI, A. **Agentes inteligentes com foco de atenção afetivo em simulações baseadas em agentes**. Universidade Federal do Rio Grande do Norte, 2012.

SILVA, E. L. d.; MENEZES, E. M. Metodologia da pesquisa e elaboração de dissertação. 3. ed. rev. atual, 2001.

SILVA, T. R. et al. **Um relato de experiência da aplicação de videoaulas de programação de jogos digitais para alunos da educação básica**. In: Anais do Workshop de Informática na Escola. [S.l.: s.n.], 2016. v. 22, n. 1, p. 141.

SIROTHEAU, S. et al. **Labpy: Laboratório virtual de ensino em pytho**n. In: Anais dos Workshops do Congresso Brasileiro de Informática na Educação. [S.l.: s.n.], 2018. v. 7, n. 1, p. 749.

SKLEARN. **SKLEARN Machine Learning in Python**. 2019. Disponível em: <https://scikit-learn.org/stable/>.

SKULPT. **Skulpt The official home of the Python Programming Language**. 2018. Disponível em: <https://github.com/skulpt/skulpt>.

SOARES, V. H. **Combinações de similaridade semântica e frequência de termos para agrupamento de textos**. Tese (Doutorado), 2017.

SOMMERVILLE, I. **Engenharia de software**, 9º edição. Pearson, Prentice Hall, v. 9, n. 9, 2011.

SQLALCHEMY. SQLALCHEMY **The Python SQL Toolkit and Object Relational Mapper**. 2018. Disponível em: <https://www.sqlalchemy.org/>.

SU, J.; SHIRAB, J. S.; MATWIN, S. **Large scale text classification using semi-supervised multinomial naive bayes**. In: CITESEER. Proceedings of the 28th International Conference on Machine Learning (ICML-11). [S.l.], 2011. p. 97–104.

SURMACZ, E. C. S.; LOPES, C. S. **A pedagogia da pergunta como estratégia didática na aula de geografia. Para Onde!?**, v. 10, n. 1, p. 99–105, 2018.

TOBAR, C. M. et al. **Uma arquitetura de ambiente colaborativo para o aprendizado de programação**. In: Brazilian Symposium on Computers in Education (Simpósio Brasileiro de Informática na Educação-SBIE). [S.l.: s.n.], 2001. v. 1, n. 1, p. 367–376.

UENO, H. Intellitutor: **A knowledge based intelligent programming environment for novice programmers**. In: IEEE. Digest of Papers. COMPCON Spring 89. Thirty-Fourth IEEE Computer Society International Conference: Intellectual Leverage. [S.l.], 1989. p. 390–395.

VICARI, R. M. et al. **Proposta brasileira de metadados para objetos de aprendizagem baseados em agentes (OBAA)**. RENOTE: revista novas tecnologias na educação [recurso eletrônico]. Porto Alegre, RS, 2010.

WATSON, C.; LI, F. W.; GODWIN, J. L. **Bluefix: Using crowd-sourced feedback to support programming students in error diagnosis and repair**. In: SPRINGER. International Conference on Web-Based Learning. [S.l.], 2012. p. 228–239.

WHITTALL, S. et al. **Codemage: educational programming environment for beginners**. In: IEEE. 2017 9th International Conference on Knowledge and Smart Technology (KST). [S.l.], 2017. p. 311–316.

WILLIAMS, L. et al. **In support of pair programming in the introductory computer science course**. Computer Science Education, Taylor & Francis, v. 12, n. 3, p. 197–212, 2002.

WOLBER, D. **App inventor and real-world motivation**. In: SIGCSE. [S.l.: s.n.], 2011.

v. 11, p. 601–606.

WOOLDRIDGE, M. **An introduction to multiagent systems**. [S.l.]: John Wiley & Sons, 2009.

YACEF, K. **Intelligent teaching assistant systems**. In: IEEE. International Conference on Computers in Education, 2002. Proceedings. [S.l.], 2002. p. 136–140.

YANG, Z. et al. **Hierarchical attention networks for document classification**. In: Proceedings of the 2016 conference of the North American chapter of the association for computational linguistics: human language technologies. [S.l.: s.n.], 2016. p. 1480–1489.

Apêndice A - Questionário de Avaliação Semanal

Avaliação - Semanal

Descrição do formulário

Nome (opcional)

Texto de resposta curta

Módulo *

1. Módulo I
2. Módulo II

Turno *

- ◯ Matutino
- ◯ Vespertino

1- Como você julga seu desenvolvimento durante as atividades desta semana? *

- ◯ SInto-me totalmente confiante para prosseguir
- ◯ Sinto-me parcialmente confiante prosseguir
- ◯ Sinto-me totalmente perdido e com muitas dificuldades
- ◯ Sinto-me parcialmente perdido e com dificuldades
- ◯ Indiferente

2 - Sobre as atividades propostas *

- Foram desafiantes, mas possíveis dentro do conteúdo ensinado e do apoio fornecido pela arquitetura pedagógica
- Foram desafiantes, mas não obtive o apoio necessário fornecido pela arquitetura pedagógica ou professor.
- Não foram desafiantes, pois eu já dominava esse conteúdo.

3 - Em comparação a abordagem tradicional de ensino, esta aplicada foi mais motivadora? *

- Discordo plenamente
- Discordo parcialmente
- Concordo Plenamente
- Concordo Parcialmente
- Indiferente

4 - Como você recebeu ajuda quando não entendeu a causa de um erro informado em seu programa? *

- Sim, totalmente através do assistente online.
- Parcialmente, o professor precisou complementar a assistência online.
- Não, somente o professor conseguiu me ajudar.
- Parcialmente, as vezes fui auxiliado pelo assistente online.
- Somente obtive ajuda com meus colegas
- Não obtive nenhuma ajuda.

5 - Quanto as respostas fornecidas pelo ambiente, você julga que: *

- Na maioria das vezes não atendeu as minhas necessidades
- Atenderam plenamente as minhas necessidades
- Atenderam parcialmente as minhas necessidades
- Não atenderam nenhuma de minhas necessidades
- Indiferente

6 - Descreva as suas dificuldades nesta semana *

Texto de resposta longa

7 - Descreva suas sugestões ou críticas sobre a metodologia utilizada. *

Texto de resposta longa

8 - Descreva suas sugestões ou críticas sobre a arquitetura pedagógica e o assistente de aprendizagem. *

Texto de resposta longa