Universidade Federal de Campina Grande
Centro de Engenharia Elétrica e Informática
Coordenação de Pós-Graduação em Informática

# Integração de um Sistema de Raciocínio Baseado em Casos e um Agente Inteligente de Diálogo para Resolução de Problemas de Programação

Gilson Pereira dos Santos Júnior

Dissertação submetida à Coordenação do Curso de Pós-Graduação em Ciência da Computação da Universidade Federal de Campina Grande - Campus I como parte dos requisitos necessários para obtenção do grau de Mestre em Ciência da Computação.

Área de Concentração: Ciência da Computação
Linha de Pesquisa: Modelos Computacionais e Cognitivos

Evandro de Barros Costa e Joseana Macêdo Fechine
(Orientadores)

Campina Grande, Paraíba, Brasil

**FICHA CATALOGRÁFICA ELABORADA PELA BIBLIOTECA CENTRAL DA UFCG**

S237i
2009

Santos Júnior, Gilson Pereira dos.
Integração de um sistema de raciocínio baseado em casos e um agente inteligente de diálogo para resolução de problemas de programação / Gilson Pereira dos Santos Júnior. ─ Campina Grande, 2009.
117 f. : il.

Dissertação (Mestrado em Ciência da Computação) – Universidade Federal de Campina Grande, Centro de Engenharia Elétrica e Informática.
Referências.
Orientadores: Prof. Dr. Evandro de Barros Costa, Profª. Drª. Joseana Macêdo Fechine.

1. Inteligência Artificial. 2. Ensino de Programação para Iniciantes. 3. Resolução de Problemas de Programação. 4. Raciocínio baseado em Casos. 5. Agente Inteligente de Diálogo. I. Título.

CDU – 004.832(043)

**"INTEGRAÇÃO DE UM SISTEMA DE RACIOCÍNIO BASEADO EM CASOS E UM AGENTE DE DIÁLOGO PARA RESOLUÇÃO DE PROBLEMAS DE PROGRAMAÇÃO"**

**GILSON PEREIRA DOS SANTOS JUNIOR**

**DISSERTAÇÃO APROVADA EM 29.06.2009**

**PROF. EVANDRO DE BARROS COSTA, D.Sc**
**Orientador**

**PROFª JOSEANA MACÊDO FECHINE , D.Sc**
**Orientadora**

**PROF. DALTON DARIO SERREY GUERRERO, D.Sc**
**Examinador**

**PROFª LELIANE NUNES DE BARROS, Drª**
**Examinadora**

**CAMPINA GRANDE – PB**

# Resumo

Aprender a solucionar problemas algoritmicamente é um dos maiores desafios no aprendizado das disciplinas de introdução à programação, uma vez que os alunos iniciantes sentem dificuldades em empregar uma das principais formas de raciocínio utilizadas pelos programadores experientes: raciocínio por meio de analogia. Para auxiliar o aluno no aprimoramento das habilidades de resolução de problemas, por meio de analogia, foi desenvolvido o ambiente *Analogus*. O *Analogus* é um ambiente de resolução de problemas de programação que visa auxiliar o aluno a identificar problemas resolvidos, similares ao atual, por meio de um raciocinador baseado em casos, ao mesmo tempo em que um agente de diálogo o auxilia a refletir sobre os aspectos de similaridade. Esse ambiente foi proposto para auxiliar na resolução das atividades práticas das disciplinas de introdução à programação, principalmente, nos cursos que seguem metodologias de ensino de padrões para alunos iniciantes. Na avaliação realizada com um grupo de alunos de graduação, da Universidade Federal de Campina Grande, foi observado e os próprios participantes concordaram que a ferramenta auxilia o aluno a lembrar de problemas semelhantes àqueles que eles estavam solucionando.

**Palavras-Chave:** Resolução de Problemas de Programação, Raciocínio por Analogias, Raciocínio Baseado em Casos, Agentes Inteligentes de Diálogo, Padrões de Programação para Iniciantes

# Abstract

Learning to solve programs using algorithms is one of the main challenges of the introductory programming courses, once beginners find difficult to use one of the most used way of thinking by expert programmers: analogy-based reasoning. In order to help beginner programmers to start using this solving approach, we have developed Analogus. It is a programming environment which helps students to identify previously solved problems which are similar to the current one, using a CBR engine alongside with a chatterbot that helps them to think about similarities aspects. This environment is recommend for solving problems in activities pratiques of programming introduction courses, especially for novice's students. In the evaluation with a students groups at Federal University of Campina Grande, was observed, and the participants agreed, that the environment helps them to students to remember similar problems.



**Keywords:** Solving Programming Problems, Analogical Reasoning, Case-Based Reasoning, Chatterbot, Novices Programming Patterns

# Dedicatória

A Josevânia, minha esposa, com amor e gratidão por sua compreensão e carinho.

# Agradecimentos

Primeiramente à Deus, pelo dom da vida e saúde no desenvolvimento deste trabalho.

A minha esposa Josevânia, pela força, dedicação, compreensão e amor, que, em todos os momentos, soube me fazer ser o homem mais feliz do mundo durante a realização deste sonho.

Aos meus pais, por todo apoio, amor, educação e ensinamentos ao longo da vida, sendo exemplos de verdadeiros pais.

Aos meus orientadores, Evandro de Barros Costa e Joseana Macêdo Fechine, pela confiança e pela dedicação na realização do trabalho.

Aos meus eternos amigos e familiares, os quais deixei em Aracaju, mas estavam sempre disponíveis para matar a saudade em todas as idas.

Aos meu grandes amigos e colega de mestrado Felipe Menezes e Fernando Henrique, pela convivência diária, pelas noites de apoio, quando a saudade da nossa terra apertava, e pelos longos anos de amizade.

Aos amigos Bruno Alexandre, Welfren, Danilo, Carlos Augusto, Guilherme, Halley, Marcos (Gaúcho), Saulo por todos os momentos de diversão.

Aos amigos e companheiros de jornada Expedito e Andréa, por todos os momentos de reflexão sobre o trabalho.

A todos que participaram do LIA (Laboratório de Inteligência Artificial), em especial a Clerton e Danielle que ajudaram na concepção do agente inteligente de diálogo.

A todos os professores do DSC, pela competência e conhecimento transmitido.

Aos funcionários da COPIN, em especial a Aninha e Vera pela presteza e simpatia.

A CAPES, pelo apoio financeiro.

# Conteúdo

# Lista de Símbolos

AIML - *Artificial Intelligence Markup Language*

ALICE - *Artificial Linguistic Internet Computer Entity*

CLIPS - *C Language Integrated Production System*

POI - *Pattern-Oriented Instruction*

RBC - *Raciocínio Baseado em Casos*

SE - *Sistema Especialista*

SELP - *Sistema para Ensino de Lógica de Programação*

SNMP - *Simple Network Management Protocol*

XML - *eXtendable Markup Language*

# Lista de Figuras

# Lista de Tabelas

# Lista de Códigos Fonte

# Capítulo 1

# Introdução

Na Seção 1.1 deste capítulo, serão contextualizadas as dificuldades enfrentadas pelos alunos iniciantes nas disciplinas de introdução à programação, evidenciando, principalmente, a sua habilidade imatura em solucionar problemas neste domínio. Em seguida, na Seção 1.2, serão apresentados os objetivos do trabalho. Na Seção 1.3 será descrita a relevância da pesquisa. Na Seção 1.4, por sua vez, serão apresentadas as contribuições da pesquisa. Por fim, na Seção 1.5, descrever-se-á a estrutura dos demais capítulos da dissertação.

## 1.1 Contextualização do Problema

O aprendizado de programação é uma tarefa complexa para alunos iniciantes, uma vez que envolve, de forma simultânea, atribuições como: (*i*) aprender uma nova linguagem, sintaxe e semântica; (*ii*) aprender um ambiente de programação; (*iii*) realizar testes; (*iv*) depurar programas e; (*v*) adquirir ou aprimorar a habilidade de resolver problemas [WB01].

Na área da psicologia, são descritos dois dos principais problemas enfrentados pelos alunos iniciantes no aprendizado de programação, quais sejam [Del05]:

- **Aprender a linguagem de programação**, que consiste em estudar a sintaxe e a semântica de uma nova linguagem;
- **Aprender a resolver problemas para o computador executar**, ou seja, o aluno aprende a solucionar problemas algoritmicamente, definindo uma sequencia de passos que devem ser executados pelo computador para solucionar o problema.

Alguns pesquisadores ressaltam que a resolução de problemas de programação é uma das tarefas de maior grau de dificuldade para o aluno [Mul05; HM08], visto que, os iniciantes

sentem dificuldades em analisar problemas e formular soluções. Contudo, evidências apontam facilidades por parte do aluno em aprender uma segunda linguagem de programação, levando à hipótese de que ao aprender uma segunda linguagem, o aluno já superou as deficiências inerentes da habilidade de resolução de problemas [Del05].

Pesquisas comprovam que a dificuldade na resolução de problemas é agravada pela habilidade imatura do aluno iniciante em raciocinar por meio de analogia [Mul05], uma vez que em algumas situações o aluno não consegue reconhecer similaridades entre problemas, recuperar soluções passadas semelhantes à atual ou, ainda, reutilizar tal solução [Mul05; MGH07].

Outro fator agravante é a ementa da disciplina de introdução à programação que, na maioria das universidades, foca no aprendizado da sintaxe e semântica de linguagens e não no desenvolvimento da habilidade de resolver problemas [Mul05].

Professores, que ministram cursos com essa característica, fornecem aos seus alunos listas de exercícios de programação que não consideram a similaridade entre os problemas em sua composição, prejudicando o desenvolvimento da habilidade do raciocínio por meio de analogia [Mul05; BRaN$^+$08a; BRaN$^+$08b].

O cenário mencionado prejudica o aprendizado dos alunos iniciantes, resultando em um elevado de índice reprovação na disciplina de introdução à programação, como também na dificuldade em acompanhar as demais disciplinas que exigem esse conteúdo como pré-requisito [OGP08].

No âmbito profissional, no entanto, é comum ver programadores experientes empregando o raciocínio por analogia no seu dia-a-dia. Eles, em inúmeras situações, recorrem a soluções passadas e as adaptam para solucionar novos problemas, evitando, dessa forma, a criação de uma solução sem informações *a priori* [BBD$^+$01; Mul05; MGH07; OGP08]. Esse conceito de reutilização empregado pelos programadores experientes é considerado uma boa prática de programação [Som06].

O caminho que leva um aluno iniciante, com a habilidade de raciocinar por meio de analogia pouco desenvolvida para resolução de problemas de programação, a se tornar um programador experiente é impregnado por dificuldades que poderiam ser mitigadas com a utilização de ferramentas e metodologias de ensino adequadas.

Ao longo dos anos, pesquisadores propuseram abordagens de ensino por meio de padrões

de programação para alunos iniciantes [CL99; Pro00; BBD+01; BFP03; MHA04; HM08]. Essa comunidade acredita que a utilização dos padrões na aprendizagem favorece o emprego do raciocínio por analogia na resolução dos problemas de programação, uma vez que, tais padrões são abstrações de soluções de um conjunto de problemas semelhantes, que podem ser instanciados para solucionar um novo problema [MHA04].

Nesse contexto, é possível citar sistemas computacionais como o ProPAT [BDM04] e o SELP [OGP08], os quais são ambientes de ensino/aprendizagem de programação por meio de padrões para iniciantes; e, também, o ambiente de ensino de lógica de programação proposto por Koslosky [Kos99] e a ferramenta de construção de abstrações em lógica de programação de Mattos [Mat00; Mat02], ambos capazes de recuperar problemas de programação similares ao atual. O estado da arte de ferramentas capazes de auxiliar o aluno iniciante no desenvolvimento do raciocínio por meio de analogia será mais detalhado no Capítulo 3.

Embora os trabalhos supracitados busquem minimizar as dificuldades apresentadas pelos alunos na resolução dos problemas de programação por meio de analogias, seja pela utilização de padrões no ensino ou pela indicação de problemas semelhantes, a literatura não relata a existência de ambientes que integrem essas duas abordagens. Tampouco, ferramentas que tenham a perspectiva de proporcionar a reflexão do aluno sobre os aspectos que tornam dois problemas semelhantes.

Desse modo, no presente trabalho ora apresentado, foi desenvolvido um ambiente virtual para resolução de problemas de programação, capaz de auxiliar o aluno iniciante a recordar problemas resolvidos, similares ao atual, ao mesmo tempo em que possibilita que o aluno reflita sobre os aspectos de semelhanças entre tais problemas.

## 1.2 Objetivos

### 1.2.1 Objetivo Geral

O principal objetivo deste trabalho é aprimorar a habilidade dos alunos de introdução à programação em resolver problemas de programação por meio do raciocínio por analogia.

Para isso, o presente trabalho se propõe a desenvolver um ambiente virtual para resolução de problemas de programação capaz de auxiliar o aluno iniciante a recordar problemas anteriormente resolvidos, similares ao atual, e fazê-lo refletir sobre os aspectos de similaridades

entre tais problemas.

### 1.2.2 Objetivos Específicos

Os objetivos específicos deste trabalho são:

1. Analisar estratégias de representação, indexação e recuperação de casos no domínio de programação;
2. Modelar e implementar um sistema de raciocínio baseado em casos para auxiliar o aluno a recordar problemas anteriormente resolvidos, similares ao atual;
3. Analisar estratégias pedagógicas de interação com aluno;
4. Modelar e implementar um agente inteligente de diálogo capaz de fazer o aluno refletir sobre os aspectos de similaridade entre os problemas recuperados e o problema atual;
5. Integrar o sistema de raciocínio baseado em casos e o agente inteligente de diálogo em um ambiente de apoio à resolução de problemas de programação.

## 1.3 Relevância

Conforme mencionado na Seção 1.1, os alunos sentem dificuldades em solucionar problemas de programação, devido à falta de habilidade em resolver problemas por meio do raciocínio por analogia. Tal situação é agravada pelo fato da ementa dos cursos não focar no desenvolvimento dessa habilidade, além da notável carência de ferramentas computacionais que proporcionem ao aluno o desenvolvimento de tal capacidade.

Diante do cenário supracitado, julga-se relevante disponibilizar à comunidade um ambiente virtual que suporte a resolução de problemas por meio de analogia e mitigue as deficiências dos alunos na resolução dos problemas de programação. Para tanto, foi desenvolvido um ambiente virtual capaz de recomendar problemas previamente resolvidos, similares ao atual, e possibilitar momentos de reflexão do aluno por meio do diálogo com um agente inteligente.

Outro aspecto considerado relevante é a integração de um sistema de raciocínio baseado em casos com um agente inteligente de diálogo, visando desenvolver o raciocínio por meio

de analogia. É possível encontrar trabalhos sobre raciocínio baseado em casos e agentes inteligentes de diálogo em aplicações nos diferentes domínios. Entretanto, não foram encontrados trabalhos que utilizam essas tecnologias em conjunto com o objetivo de aprimorar a habilidade de resolução de problemas de programação.

Além disso, destaca-se a relevância do estudo de uma forma adequada de indexação, representação e recuperação de casos no domínio de programação, buscando recuperar de maneira eficiente problemas de programação, cujas soluções sejam úteis para serem reutilizadas na resolução de problemas semelhantes.

Vale ainda ressaltar, que introduzir os conceitos de reuso nas disciplinas introdutórias de ciência da computação favorecerá na obtenção, ao final do curso, de bons solucionadores de problemas e, consequentemente, bons profissionais de informática. Com isso, será possível reduzir os números de reprovação elevados nas disciplinas de programação e encaminhar ao mercado de trabalho profissionais bem mais preparados.

## 1.4 Contribuições

O presente trabalho contribui diretamente em duas áreas: inteligência artificial e educação em ciência da computação.

Na área de inteligência artificial, a pesquisa contribui com o estudo de formas adequadas de indexação, representação e recuperação de casos no domínio de programação. Como também, com a integração entre um raciocinador baseado em casos e um agente inteligente de diálogo.

No contexto da educação em ciência da computação, o trabalho contribui com a disponibilização de um ambiente virtual para ensino das disciplinas de introdução à programação, principalmente em cursos cuja ementa foca no desenvolvimento da habilidade de resolver problemas, por meio do ensino de padrões de programação para iniciantes, não apenas no aprendizado da sintaxe e da semântica de linguagens.

Ainda nessa última área, o emprego do agente na reflexão dos aspectos de similaridades entre os problemas poderá contribuir com o aprimoramento da habilidade do aluno em identificar problemas semelhantes.

## 1.5 Estrutura da Dissertação

O restante da dissertação está organizada em 5 (cinco) capítulos.

No **Capítulo 2** está apresentada a fundamentação teórica, contendo os principais conceitos técnicos utilizados na pesquisa, buscando descrever o estado da arte do tema tratado.

No **Capítulo 3** são abordadas as principais ferramentas de auxílio aos alunos iniciantes no desenvolvimento do raciocínio por meio de analogia para resolução de problemas de programação.

No **Capítulo 4**, é descrito, de forma detalhada, o ambiente virtual de resolução de problemas de programação por meio do raciocínio por analogia.

No **Capítulo 5**, por sua vez, é descrita metodologia de avaliação e, em seguida, são apresentados e analisados os resultados obtidos.

Por fim, no **Capítulo 6**, são apresentadas as considerações finais da pesquisa, bem como, as sugestões para trabalhos futuros.

# Capítulo 2

# Fundamentação Teórica

Neste capítulo, são apresentados conceitos básicos relacionados à pesquisa, os quais são fundamentais para o melhor entendimento da mesma. Para tanto, esse capítulo está subdividido em seções. Na Seção, 2.1 será descrito o raciocínio por meio de analogia, bem como a forma como os seres humanos empregam essa heurística na resolução de problemas. Já na Seção 2.2, serão apresentados os conceitos de raciocínio baseado em casos. Em seguida, na Seção 2.3 serão tratados os padrões de programação para alunos iniciantes. Por fim, na Seção 2.4, serão apresentados os conceitos necessários para a fundamentação teórica sobre agentes inteligentes de diálogo.

## 2.1 Raciocínio por meio de Analogia

O raciocínio por meio de analogia é um mecanismo cognitivo que possibilita entender uma nova situação mediante o conhecimento extraído de experiências semelhantes anteriormente vivenciadas [Mul05]. Esse mecanismo é uma das principais habilidades do raciocínio humano, o que o torna, ainda hoje, um resolvedor/solucionador de problemas mais eficiente do que qualquer ferramenta computacional da inteligencia artificial [Mul05].

Nesse contexto, as experiências passadas são chamadas de origem da analogia e fornecem indícios de como inferir sobre uma situação não familiar, o alvo da analogia [Mul05].

O raciocínio por meio de analogia pode ser descrito em 4 (quatro) etapas [Mul05]:

1. Identificar na memória situações análogas à situação não familiar;

2. Mapear os esquemas cognitivos das experiências passadas no alvo da analogia;

3. Inferir conhecimento sobre o alvo mediante os esquemas das situações familiares;

4. Compreender melhor a nova situação a partir das inferências realizadas.

As dificuldades em solucionar problemas por meio de analogias, geralmente, são provenientes da falha em um dos passos supracitados. Por isso, é fundamental saber recuperar da memória os problemas similares, mapear esquemas cognitivos, inferir informações corretas sobre eles e, por fim, conseguir compreender a nova situação [Mul05].

Por outro lado, desde a sua infância, o homem exercita essa forma de raciocínio, visto que, é comum observar crianças, no decorrer do seu aprendizado, associando cores e formas de objetos conhecidos aos novos que estão sendo visualizados [Gos93].

Segundo Vygotsky (1934) *apud* [Poz02], aprender um novo conceito é semelhante a aprender uma língua estrangeira. No aprendizado de uma nova língua, é natural os seres humanos se apoiarem na língua materna. O mesmo ocorre no aprendizado de um novo conceito, considerando que toda aprendizagem tem um conhecimento prévio que pode ser útil.

Educadores exploram e pregam a analogia como uma alternativa de raciocínio que auxilia o aprendiz a elaborar conhecimento quando este está lidando com conhecimentos complexos em diferentes conteúdos [NDJ01].

Glynn [Gly07] introduziu o modelo *"Teaching with Analogies (TWA)"*. Esse modelo de ensino por meio de analogias é constituído por 6 (seis) passos, sendo esses:

1. Introduzir o conceito alvo;

2. Recordar conceitos análogos;

3. Identificar características similares entre tais conceitos;

4. Mapear as características similares;

5. Esboçar uma conclusão sobre o conceito;

6. Indicar os limites da analogia.

Embora a analogia deva ser considerada um processo criativo, é muito importante saber reconhecer os seus limites, evitando, dessa forma, inferência de conceitos errados [NDJ01].

A vantagem do aprendizado por meio de analogia está associada ao fato de que é mais fácil compreender um caso concreto do que uma abstração, independente do nível escolar

do estudante, visto que o entendimento adquirido com o caso específico pode ser transferido para uma nova situação [GLT03].

Atualmente, essa forma de ensino é amplamente utilizada nas universidades em cursos como administração, matemática, medicina e direito [GLT03]. Por exemplo, Polya [Pol71] sugere que é usual a busca por situações análogas na resolução de problemas no domínio da geometria.

Segundo Polya [Pol71], um aluno, após algumas experiências com problemas semelhantes, poderá perceber as ideias básicas para a resolução de um problema: a utilização dos dados relevantes, a variação dos dados, a simetria, a analogia. Com esse hábito, o aluno poderá melhorar sua capacidade de resolver problemas.

Na computação, alguns pesquisadores incentivam o emprego dessa metodologia no ensino de programação, como poderá ser visto com mais detalhes na Seção 2.3.

## 2.2 Raciocínio Baseado em Casos

O Raciocínio Baseado em Casos (RBC), do inglês *Case-based Reasoning*, é uma abordagem de resolução de novos problemas por meio da adaptação de soluções de problemas similares anteriormente resolvidos, proposta pela Inteligência Artificial [WM94].

Para isso, o raciocinador baseado em casos busca em sua memória de casos (base de casos) problemas com descrições semelhantes ao problema atual e, a partir da solução dos problemas similares recuperados, a solução do novo problema é gerada e devolvida pelo raciocinador [SP04a]. Na Figura 2.1, está ilustrado esse processo. Maiores detalhes sobre o funcionamento do Raciocínio Baseado em Casos estão descritos na Seção 2.2.2.

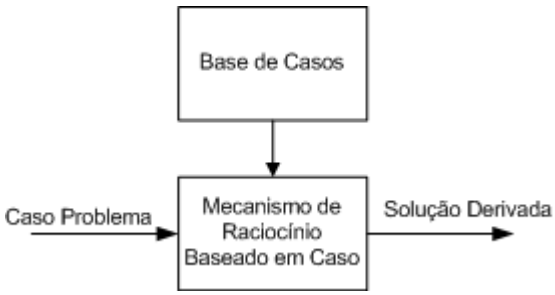



Figura 2.1: Visão Geral do RBC. Adaptado de [SP04a].

A forma de raciocínio apresentada pelo RBC pode ser vista como um caso particular

do raciocínio por meio de analogia, isso porque no raciocínio por analogia o conhecimento adquirido, a partir de situações passadas podem ser utilizadas para solucionar problemas de outros domínios, enquanto que no raciocínio baseado em casos o conhecimento é empregado no mesmo domínio das situações anteriores. Vale ressaltar que essa forma de raciocínio é amplamente utilizada pelos seres humanos no seu cotidiano [SP04a].

Desse modo, os estudos sobre o raciocínio baseado em casos têm raízes em diferentes disciplinas como, por exemplo, as ciências cognitivas, a representação e processamento de conhecimento, a aprendizagem de máquina e a matemática [RA05].

Assim, é importante destacar que a comunidade de Inteligência Artificial se espelhou nessa capacidade humana de raciocinar por meio de analogia para o desenvolvimento do RBC, como pode ser observado em sua história, na Seção 2.2.1.

### 2.2.1 História do RBC

Os trabalhos de Schank e Abelson sobre memória dinâmica e o registro de situações padrão por meio de *script*, em 1977, foram o marco inicial das pesquisas em raciocínio baseado em casos [AP94; dMMB$^+$05].

Na área das ciências cognitivas, pesquisas buscando: (*i*) entender a aprendizagem de uma nova habilidade pelas pessoas e (*ii*) observar como os seres humanos geram hipóteses sobre uma nova situação baseados em situações passadas, impulsionaram os estudos sobre RBC. O objetivo dessas pesquisas era construir um sistema de suporte à decisão capaz de auxiliar as pessoas em seu aprendizado.

Inspirada por tais pesquisas, Janet Kolodner, da Universidade de Yale, lançou em 1983 o primeiro sistema que utiliza raciocínio baseado em casos, denominado CYRUS. O CYRUS é um sistema de perguntas e respostas constituído pelas viagens e reuniões diplomáticas de Cyrus Vance, ex-secretário de Estado dos Estados Unidos, descrito em forma de caso e implementado com *MOPs* (*Memory Organization Packet*)[1]. Posteriormente, o modelo de casos empregado no CYRUS serviu de base para outros sistemas como, por exemplo: o Mediator, utilizado para resolução de consenso entre partes; o Chef que gera novas receitas a partir de outras e; o Persuader, sistema que resolve conflitos entre patrões e empregados

[1]Estrutura de organização em memória, criada por Shank em 1982, que armazena o conhecimento sobre um tipo de evento, organizado individualmente em episódios.

[AP94].

Em 1988, 1989, e 1991 a *U.S. Defense Advanced Research Projects Agency* (DARPA) organizou os 3 (três) primeiros *workshops* sobre raciocínio baseado em casos. Hoje, a linha de pesquisa de RBC está presente nos principais eventos de Inteligência Artificial como, por exemplo, o *European Conference on Artificial Intelligence* (ECAI) e o *International Joint Conference on Artificial Intelligence* (IJCAI), além das conferências especializadas em RBC como a *International Conference on CBR* (ICCBR) e a *European Conference on CBR* (ECCBR) [SP04a].

Os avanços do RBC não ficaram apenas no âmbito acadêmico, uma vez que com o passar dos anos, o interesse crescente pela área levou ao desenvolvimento de ferramentas comerciais como o *CBR Express* produzido pela *Inference Corporation* e o *ReMind* da *Cognitive System Inc*. Atualmente, é possível encontrar sistemas de raciocínio baseado em casos no domínio da educação, direito, engenharia, dentre outros [AP94].

### 2.2.2 Funcionamento do RBC

Ao longo dos anos, alguns trabalhos apresentaram diferentes arquiteturas de sistemas de raciocínio baseado em casos [RA05]. Porém, em 1994, Aamodt e Plaza definiram as principais características de um RBC em um modelo de processo simples e uniforme, constituído por 4 (quatro) fases, sendo essas [AP94]:

- **Recuperação:** recupera da base um conjunto de casos com descrição do problema similar ao problema consultado;
- **Reuso:** reutiliza as soluções dos casos recuperados para solucionar o problema atual;
- **Revisão:** revisa a solução proposta;
- **Retenção:** aprende a nova experiência para solucionar problemas futuros.

De acordo com o ciclo proposto por Aadmont e Plaza, ilustrado na Figura 2.2, a descrição de um problema define um novo caso. Assim, quando o novo caso é apresentado ao raciocinador, o RBC recupera uma coleção de experiências passadas, cujas soluções (desses casos recuperados) são utilizadas para gerar uma solução inicial para o novo problema. Caso exista um problema idêntico ao atual, dentre os casos recuperados, a solução deste será a solução do

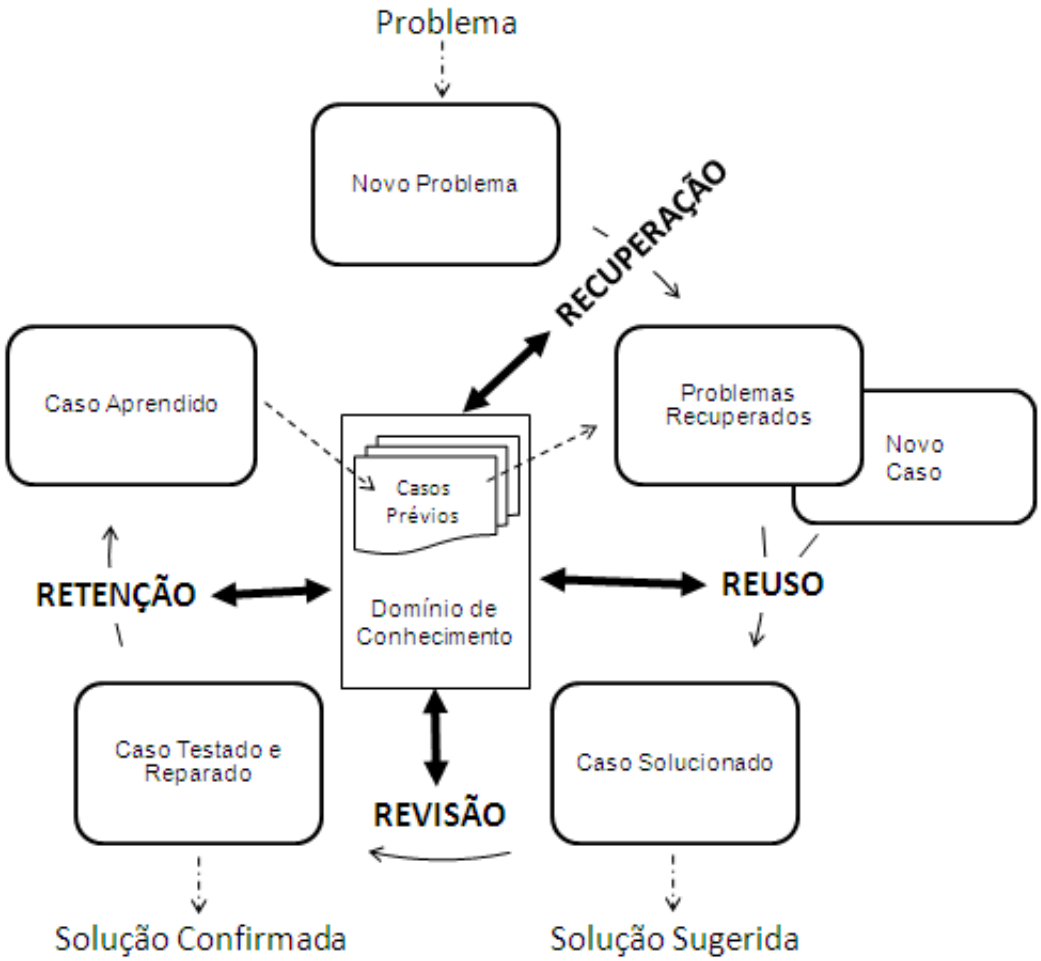



Figura 2.2: Ciclo clássico do RBC. Adaptado de [SP04a].

novo caso. Por outro lado, em situações em que isso não acontece, a solução do novo caso se dá por meio da adaptação da solução dos casos recuperados. Em seguida, a solução gerada é revisada e testada no ambiente real, efetuando-se os ajustes necessários e, logo que a solução esteja confirmada, o novo caso é armazenado para ser reutilizado futuramente [AP94].

Após a definição do ciclo básico proposto por Aamodt e Plaza, alguns trabalhos foram criados refinando esse modelo, adicionando outros elementos ao ciclo ou dividindo as fases em subciclos [RA05].

Em algumas situações, o ciclo do raciocínio baseado em casos é incapaz de solucionar o problema por não possuir na memória de casos uma situação similar à consultada e, portanto, criar uma solução do zero torna-se mais vantajoso do que adaptar um solução a partir dos casos existentes.

### O Caso

O caso é um trecho de um conhecimento contextualizado, devido a uma experiência ou situação passada, que fornece subsídios para solucionar um problema presente [WM94].

Esse registro é uma abstração do mundo real e, geralmente, é constituído por:

- **Problema:** descreve o cenário em que ocorreu o caso.
- **Solução:** registra as medidas tomadas para solucionar o problema, o conjunto de passos para atingir a solução e o estado do mundo após a solução.

O caso pode ser representado por inúmeras formas, dentre as quais: relacional, frames, redes semânticas e objetos. A representação relacional é simples, flexível e é a mais adotada em sistemas comerciais, de modo que cada caso pode ser representado por uma linha em uma tabela relacional, enquanto as colunas representam os atributos deste caso.

No momento da escolha da forma de representação do caso, alguns fatores devem ser considerados, como [SP04a]:

- **Estrutura interna do caso:** a representação escolhida deve ser adequada à estrutura interna do caso como, por exemplo, suportar os tipos de dados utilizados;
- **Indexação e recuperação:** a representação deve ser adequada à forma de indexação e mecanismo de busca escolhidos;
- **Linguagem ou Shell:** a linguagem de consulta dos casos ou Shell utilizado pode limitar à forma de representação dos casos.

A representação do caso, de modo geral, pode ser vista como um conjunto de pares (atributo-valor) indexados ou não. Os atributos não indexados são utilizados para armazenar informações sobre o contexto do caso, mas não auxiliam na recuperação. Já os atributos indexados, também conhecidos por índices, armazenam informações que auxiliam na redução do espaço de busca de um caso, tornando o mecanismo de recuperação mais eficiente [CBR03].

Em um sistema de recuperação de problemas de programação, os padrões de programação, a categoria e a complexidade do problema são exemplos de atributos indexados; enquanto que o nome do aluno e sua foto são atributos não indexados nesse contexto.

É fundamental ressaltar a importância da escolha dos atributos indexados, pois, estes estão diretamente relacionados à precisão e à velocidade de recuperação do caso pelo mecanismo de busca. Desse modo, os índices devem refletir características importantes do caso como, por exemplo, a circunstância em que o caso ocorre [SP04a].

Para um sistema de raciocínio baseado em casos, um bom índice é aquele que consegue ser distintivo e não unívoco, simultaneamente [SP04a]. Por exemplo, o atributo cor do carro é um bom índice, isso porque, a partir da cor é possível distinguir os carros (distintivo), ao mesmo tempo em que esse atributo não representa uma única ocorrência, ou seja, existe um grupo de carros com a determinada cor (não unívoco).

Segundo Kolodner, um índice de qualidade deve [Kol93]:

- prever a futura utilização da informação para solucionar diferentes problemas;
- endereçar as similaridades úteis entre os casos;
- ser abstrato o suficiente para tornar o caso útil em diferentes situações;
- ser concreto o suficiente para facilitar o seu reconhecimento em futuras situações.

Na fase de recuperação, esses índices são utilizados no cálculo da similaridade entre os casos, como será visto a seguir, na recuperação do caso.

### Recuperação do Caso

A fase de recuperação é o processo pelo qual são obtidos um ou mais casos da base, similares ao consultado. Um mecanismo de recuperação eficiente deve saber julgar quais casos na base possuem informações relevantes para solucionar o novo problema.

Diante disso, a recuperação pode ser dividida em 3(três) etapas [AP94]:

- **Assessoria na consulta:** auxiliar o usuário na criação da consulta com os atributos relevantes para recuperação do caso;
- **Casamento:** recuperar um conjunto de casos similares ao caso consultado para solucionar o problema;
- **Seleção:** identificar o melhor caso dentre os selecionados no casamento. Nessa etapa, os casos obtidos no casamento são ordenados pelo grau de similaridade de acordo com um critério de seleção. Por fim, o mais similar é selecionado.

O algoritmo de recuperação de casos depende da forma pela qual o referido caso está armazenado e indexado. Assim, dentre os métodos mais conhecidos, estão o algoritmo do vizinho mais próximo (*Nearest Neighbour*), o algoritmo de indução e o de recuperação de padrões [SP04a]. Neste documento foi detalhado o *Nearest Neighbour* por ter sido utilizado na pesquisa.

O algoritmo do vizinho mais próximo consiste na comparação entre o caso consultado e os casos armazenados na base. Tal comparação ocorre por meio da similaridade local e global.

A similaridade global é dada pela média ponderada das similaridades locais. Já a similaridade local, é a função que informa o quão semelhantes são os valores para um determinado atributo. E, para cada atributo é associado um peso que representa o grau de importância daquele atributo no domínio tratado.

Devido a sua importância, o ajuste dos pesos pode ser efetuado a partir do conhecimento de um especialista do domínio ou por um processo de aprendizagem adaptativa. Alguns trabalhos apontam a utilização de algoritmos genéticos nesse ajuste [SP04b].

Formalmente, o cálculo da similaridade global é obtido a partir da equação (Equação 2.1):

$$Sim(C, R) = \frac{\sum_{i=1}^{n} w_i * f(C_i, R_i)}{\sum_{i=1}^{n} w_i} \tag{2.1}$$

em que:

$C$ = caso consultado;

$R$ = caso recuperado da base;

$n$ = número de atributos no caso;

$w$ = peso atribuído ao *i-ésimo* atributo;

$i$ = *i-ésimo* atributo;

$f$ = função que calcula a similaridade local entre os casos $C$ e $R$ para o *i-ésimo* atributo.

A similaridade local depende diretamente do tipo de dado do atributo. Por exemplo, um caso contendo um atributo inteiro pode ser considerado similar a outro quando ambos possuírem o mesmo valor ou quando a diferença entre eles está abaixo de um limiar. Ressalta-se, que os esforços em encontrar funções de similaridade local adequadas para cada tipo de dado

resultou no desenvolvimento de inúmeras funções, cujas principais são [SP04a]:

- **Distância Euclidiana:** consiste na medida da distância entre dois objetos no espaço euclidiano. O cálculo da distância é calculado a partir da raiz quadrada do quadrado da diferença aritmética entre duas coordenadas no espaço euclidiano, ajustado pelo peso. Nessa equação, o peso indica a importância do atributo no caso. Formalmente, o cálculo se dá como apresentado na Equação 2.2.

$$d_{pq}^{w} = \sqrt{w^2(x_p - x_q)^2} \tag{2.2}$$

  em que:
  $d_{pq}^{w}$ = é a distância euclidiana entre as coordenadas $p$ e $q$, ajustada pelo peso $w$.

  Essa medida de distância é comumente utilizada para identificar similaridades entre os atributos do raciocinador baseado em casos.

- **Distância de *Hamming*:** calcula a distância a partir do número de *bits* diferentes na comparação entre dois vetores. Por exemplo, seja $p$ = 01001 e $q$ = 11000, a distância $d(p, q)$ = 2. Essa função foi desenvolvida inicialmente para detecção e correção de erros em comunicação digital. No contexto de RBC, o caso mais similar ao consultado é aquele que possui a menor distância de *hamming*.

- **Distância de *Levenshtein*:** consiste no cálculo do número de transformações - deleção, inserção ou substituição - realizadas em uma *string* para transformá-la em outra, em que cada transformação está associada a um custo e a *string* similar à consultada é aquela que possui o menor custo total.

- ***Cosine Similarity*:** medida de similaridade entre 2 (dois) vetores dada pelo coseno do angulo entre eles. Essa medida é comumente usada para comparar documentos textuais, nos quais o vetor é constituído pela frequência dos termos existentes no documento. Supondo que A e B sejam vetores de tamanhos distintos.

$$Sim(A, B) = \frac{A.B}{\sqrt{\mid A \mid} * \sqrt{\mid B \mid}} \tag{2.3}$$

- ***Jaccard Coefficient*:** computa a relação entre a similaridade e diversidade de um conjunto. O cálculo é dado pela divisão da quantidade de elementos da interseção pela quantidade de elementos da união. Na Equação 2.4, está descrita essa função.

$$Sim(A, B) = \frac{A \cap B}{A \cup B} \tag{2.4}$$

- ***Dice's Coefficient*:** medida similar ao Jaccard, porém adiciona o conceito de bigrama. Os bigramas são grupos de duas letras, por exemplo, a palavra livro possui os bigramas = li, iv, vr, ro.

$$Sim(A, B) = \frac{2 \mid A \cap B \mid}{\mid A \mid + \mid B \mid} \tag{2.5}$$

- ***Overlap Coefficient*:** relação entre o tamanho da interseção do conjuntos A e B, e o tamanho do menor conjunto (A ou B). Essa relação indica o quão próximo o menor conjunto está de ser um subconjunto do conjunto maior.

$$Sim(A, B) = \frac{\mid A \cap B \mid}{min(\mid A \mid, \mid B \mid)} \tag{2.6}$$

Diante do cálculo da similaridade entre os casos é fácil perceber que, a depender da função escolhida e do tamanho da base de casos, o custo computacional pode ser elevado. Por isso, é importante ressaltar que, diferentemente dos bancos de dados convencionais que recuperam registro com campos idênticos ao consultado, o raciocínio baseado em casos recupera casos a partir de um julgamento de atributos, retornando casos com atributos que podem possuir informações faltantes ou que não casem perfeitamente. É notável que esse processo é bem mais complexo do que o realizado pelos bancos de dados convencionais, o que torna esta uma limitação do RBC quando a base de dados a ser consultada possui um volume muito grande de casos.

Uma vez recuperados os casos similares, o sistema de raciocínio baseado em casos entra na fase de reuso.

### Reuso

Esta fase consiste em reutilizar as soluções dos casos recuperados para solucionar o problema corrente, efetuando as devidas adaptações, se necessário [dMMB+05].

Desse modo, o reuso foca em 2 (dois) aspectos:

1. Nas diferenças entre os casos recuperados e o corrente;
2. Nas informações que podem ser transferidas dos casos recuperados para o atual.

Normalmente, os casos recuperados não casam perfeitamente com a descrição do problema consultado. Então, é necessário efetuar a adaptação da solução dos casos recuperados para solucionar o novo problema. Tal adaptação pode ser efetuada por meio de um dos métodos detalhados a seguir [SP04a].

- **Reinstanciação:** a solução de um dos casos recuperados é copiada na íntegra para o novo problema.
- **Substituição:** a solução é gerada a partir dos casos recuperados, tratando os conflitos ou contradições dos atributos por meio da substituição dos seus valores, a fim de atender os requisitos do novo caso.
- **Transformação:** a solução é derivada de regras e características dos requisitos do novo caso. Essa forma de adaptação é comumente utilizada quando a substituição não pode ser efetuada.

Uma vez completada a adaptação, o caso está pronto para ser testado no ambiente real, visando verificar se a solução desenvolvida está adequada ao problema. Essa etapa é realizada na fase de revisão.

#### Revisão

Assim que a solução é gerada, na fase de reuso, esta é testada no ambiente real para verificar se a reutilização foi efetuada adequadamente [AP94]. Esta fase de revisão consiste em 2 (duas) tarefas:

- **Avaliar a solução** gerada na fase de reuso, identificando as possíveis falhas;
- **Reparar as falhas** da solução, para adequá-la ao ambiente real.

Após avaliar a solução e efetuar os reparos necessários, o ciclo do RBC chega a fase de retenção, na qual ocorre a aprendizagem do sistema.

#### Retenção

A fase de retenção consiste no processo de aprender o conhecimento gerado, permitindo sua utilização futura para solucionar um novo problema [dMMB+05]. Essa fase é caracterizada por 3 (três) tarefas, são essas:

- **Extração:** extrair do RBC a informação de como o caso foi solucionado. Tal informação é armazenada juntamente com o problema e a solução do caso, no intuito de fornecer pistas de como solucionar novos casos similares a este.

- **Indexação:** gerar os índices necessários para o armazenamento do novo caso. Essa etapa é fundamental, uma vez que nela são decididos quais índices serão utilizados no caso, com o intuito de facilitar sua posterior recuperação e determinar como o novo caso será estruturado no espaço de busca. É importante ressaltar, que essa etapa está diretamente relacionada à representação e indexação do caso.

- **Integração:** consiste na etapa final da retenção. Nela ocorre a concretização do armazenamento do caso.

Uma vez que o caso foi recuperado, adaptado, revisado e aprendido pelo RBC, o ciclo se encerra até que um novo caso seja consultado.

### 2.2.3 Quando utilizar o RBC

O raciocínio baseado em casos pode ser empregado para solucionar diversos tipos de problemas, dos mais diferentes domínios. Contudo, para justificar sua utilização, é importante observar se o problema atende aos 5 (cinco) requisitos apresentados a seguir, na forma de questionamentos [SP04a; SP04b].

- **É impossível compreender perfeitamente o domínio?** É impossível compreender perfeitamente o domínio ou modelar os critérios que levam ao sucesso ou insucesso da solução do problema?

- **Existem exceções e novos casos?** Sistemas sem exceções ou novos casos devem ser modelados com regras, uma vez que é possível determinar previamente todas as possibilidades. Entretanto, situações em que ocorrem frequentemente exceções e novos

casos, torna-se difícil manter um sistema de regras, incentivando, portanto, o uso do RBC que aprende incrementalmente diante da inclusão de um novo caso.

- **A situação é recorrente?** As situações representadas nos casos são recorrentes ao ponto de existirem na base de casos situações similares à consultada?

- **É vantajoso adaptar uma situação passada?** É vantajoso construir a solução de um novo problema a partir da adaptação de uma solução passada ao invés de criar uma solução para um problema a partir do zero?

- **As situações passadas fornecem informações relevantes para solucionar o novo problema?** Os casos passados possuem características relevantes do problema e do contexto em que ele ocorre? As soluções possuem detalhes suficientes para gerar a solução de um novo problema por meio da adaptação?

Caso os questionamentos apresentados sejam respondidos positivamente, é recomendado o desenvolvimento de um sistema de raciocínio baseado em casos. Do contrário, será mais adequado buscar outras alternativas como, por exemplo, um sistema de raciocínio baseado em regras.

Assim, destaca-se que no desenvolvimento de um sistema para auxiliar o aluno na identificação de problemas de programação similares é possível responder positivamente a esses questionamentos. Destaca-se, por exemplo, os programadores experientes que adaptam soluções dos problemas já resolvidos para solucionar novas situações, provando a relevância das informações passadas na solução de novos problemas e que adaptar é uma prática vantajosa. Além disso, é difícil para os especialistas descreverem algoritmicamente todos os problemas de programação, embora possam ser criados alguns exemplos que representam grupos de problemas, como é o caso dos padrões.

### 2.2.4 Vantagens do RBC

O raciocínio baseado em casos apresenta algumas vantagens, sendo estas [SP04a; SP04b]:

- **Redução de esforço na tarefa de aquisição de conhecimento:** No RBC a tarefa de aquisição de conhecimento consiste em armazenar um conjunto de experiências/casos,

diferente dos sistemas baseados regras[2] que precisam extrair um conjunto de regras do especialista;

- **Evitar erros cometidos no passado:** É possível registrar informações sobre as causas de falhas nos casos passados para evitar futuras falhas;
- **Flexibilidade na modelagem do conhecimento:** Não é necessário modelar completamente o domínio a *priori*, uma vez que o RBC se utiliza de experiências passadas e é capaz de fornecer uma solução razoável mesmo sem conhecer o domínio perfeitamente;
- **Aprendendo com o tempo:** O RBC aprende quando se depara com novos problemas nos quais as soluções são geradas, testadas e armazenadas. Com os casos adicionados o sistema pode raciocinar a partir de mais situações, aumentando o grau de refinamento e sucesso da solução;
- **Raciocínio com dados incompletos ou imprecisos:** O RBC é capaz de recuperar casos não exatamente idênticos e, a partir deles, criar uma solução aceitável. Essa habilidade do raciocínio baseado em casos permite recuperar casos com informações relevantes para solução sem a necessidade de ter dados corretos e precisos como entrada;
- **Extensível para diferentes domínios:** O raciocínio baseado em casos pode ser estendido e aplicado em diferentes domínios. Ele pode ser desenvolvido para criação de planos, identificação de diagnósticos, recomendação de produto e argumentação de um ponto de vista [SP04a];
- **Reflete o raciocínio humano:** O RBC imita a forma de raciocínio por meio de analogia empregada pelos seres humanos.

### 2.2.5 Discussão

A flexibilidade do raciocínio baseado em casos possibilita seu emprego nos mais diferentes domínios de aplicação, podendo funcionar como um resolvedor de problemas ou até mesmo

[2]trata-se de sistemas que utilizam regras explícitas para expressar o conhecimento de um especialista sobre o domínio de um problema e permite, através da confrontação do conhecimento existente com fatos conhecidos sobre um determinado problema, inferir regras relativas a esses fatos.

como um recomendador de produtos.

A área de raciocínio baseado em casos possibilita, ainda hoje, pesquisas promissoras como, por exemplo: a aplicação do RBC em novos domínios; o estudo de novas formas de representar o caso; o desenvolvimento de métricas de similaridade e; mecanismos de busca mais eficientes.

Além disso, ainda há o emprego do RBC em conjunto com outras técnicas de Inteligência Artificial como a Lógica Fuzzy, utilizada na produção de regras para guiar a adaptação dos casos; as Redes Neurais, para resolução de problemas complexos e na recuperação de casos e; os Algoritmos Genéticos, no ajuste de pesos da similaridade local e global [SP04b].

Uma subárea do RBC ainda bastante promissora é a pesquisa de sistemas de raciocínio baseado em casos textuais, os quais têm o objetivo de recuperar casos textuais relevantes para solucionar problemas de uma forma geral, extrair ou destacar trechos relevantes nos textos, ou raciocinar sobre os casos para interpretar um problema [WAB05].

Na área de educação, em cursos de resolução de problemas ou desafios, os alunos podem acessar bibliotecas de casos, na qual são encontrados exemplos que podem ajudar o aluno ao chegar no caminho da resolução do problema, como é o caso do *Goal-Based Scenarios* e o *Learning by Design* [KCGC05].

## 2.3 Ensino de Programação Baseado em Padrões

O ensino de programação apresenta dificuldades tanto para os alunos, que precisam aprimorar a habilidade de resolução de problemas e aprender a sintaxe e semântica de uma linguagem de programação, quanto para os professores que devem auxiliá-los nesse aprendizado.

Diante dos problemas relatados na Seção 1.1, a comunidade de ensino de programação buscou, durante anos, alternativas para auxiliar o professor no ensino da disciplina de introdução à programação, visando maximizar o aprendizado do aluno e melhorar o seu desempenho no curso.

Com base em tais estudos, a comunidade da área de Padrões Pedagógicos desenvolveu uma metodologia de ensino baseada na instrução de padrões de programação, a *Pattern-based Programming Instruction*.

Os padrões são abstrações de trechos de código, que descrevem a solução para um con-

junto de problemas análogos e, também são bons exemplos de soluções elegantes e eficientes desenvolvidas por especialistas [MHA04].

Geralmente, um dado problema de programação exige em sua solução a combinação de dois ou mais padrões. A combinação de padrões pode ser do tipo [MGH07]:

- **Sequenciamento:** os padrões são colocados sequencialmente, um após o outro;
- ***Merge*:** um padrão contém o outro. Em outras palavras, para alcançar a solução do problema é necessário que seja inserido no escopo de um padrão, o outro padrão.

A combinação de padrões define o grau de complexidade do problema solucionado. Pesquisas apontam que os alunos apresentam mais dificuldades em realizar a combinação por meio do *merge* de padrões [MGH07].

Os padrões de programação devem ser documentados por meio de um *template*, o qual registra decisões tomadas, alternativas e consequências do uso do padrão. Gamma *et. al* [GHJV95] sugere um *template* para descrição de padrões de projeto, conforme ilustrado na Tabela 2.1.

Tabela 2.1: *Template* da descrição de padrões de projeto.

| **Elemento** | **Descrição** |
|---|---|
| Nome do Padrão | Termo que referencia o padrão (problema, solução e consequência). |
| Intenção | O que o padrão faz? |
| Nomes Alternativos | Outros nomes conhecidos do padrão. |
| Motivação | Um cenário que ilustra um problema e como este é resolvido pelo padrão. |
| Aplicabilidade | Em quais situações o padrão pode ser aplicado? |
| Estrutura | Representação gráfica do padrão. |
| Participantes | Classes e/ou objetos utilizados no padrão e suas responsabilidades. |
| Colaborações | Como os participantes colaboram. |
| Consequências | Quais os resultados do emprego do padrão? |
| Implementação | Conhecimento técnico necessário para implementar o padrão. |
| Código de Exemplo | Fragmentos de código da implementação do padrão em uma linguagem. |
| Usos Conhecidos | Exemplos do uso do padrão em sistemas reais. |
| Padrões Relacionados | Quais padrões estão relacionados a este? Quais suas diferenças? |

Os padrões de programação empregados no ensino de alunos iniciantes são categorizados em [MHA04]:

- **Padrões elementares:** são simples padrões de projeto que podem ser utilizados por iniciantes no aprendizado de programação. Essa categoria de padrões foca na estrutura, sintaxe e se refere a operações básicas como seleção, repetição, funções, etc;

- **Padrões algorítmicos:** são blocos que representam a base para soluções de problemas. Os algorítmicos são padrões que focam na semântica do problema e representam soluções básicas de busca de itens e identificação de valores extremos [MHA04].

Mais detalhes sobre os padrões elementares e algorítmicos serão descritos a seguir, nas Seções 2.3.1 e 2.3.2, respectivamente.

Durante anos, o ensino de programação, por meio de padrões para alunos iniciantes, foi debatido em eventos como o *Chili PLoP*, o *Symposium on Computer Science Education* (SIGCSE), o *Pattern Languages of Programs Conference* (PLoP), o *Consortium for Computing Sciences in Colleges* (CCSC) e o *Educators' Symposium* no OOPSLA.

Em 2001, o CCSC publicou o posicionamento dos principais pesquisadores da área sobre a utilização de padrões nos cursos introdutórios de programação. Neste trabalho, é possível encontrar relatos das experiências de Joseph Bergin, Alyce Brady, Robert Duvall, Richard Rasala e Viera Proulx [BBD+01].

### 2.3.1 Padrões Elementares

Os padrões elementares são trechos de código que representam padrões simples de projeto para alunos iniciantes. Esses padrões focam na estrutura da linguagem de programação. Logo, devem ser simples e concisos [Del05].

Tais padrões apresentam algumas vantagens no aprendizado do aluno iniciante como, por exemplo [Del05]:

- **Facilidades no aprendizado da linguagem de programação:** o conjunto de exemplos e indicações de quando utilizá-lo, o qual é encontrado na documentação dos padrões, ajuda o aluno a perceber detalhes da linguagem estudada;

- **Facilidade na comunicação entre o professor e o aluno:** o vocabulário de soluções de problemas facilita a comunicação entre o professor e o aluno iniciante criando um dialeto que é entendido por ambos.

Os padrões elementares são bastante úteis para alunos iniciantes, principalmente, nas primeiras semanas de curso, momento em que é necessário aprender elementos básicos da linguagem, estruturas de seleção, estrutura de repetição e outros.

Padrões de seleção simples, repetição com contador ou sentinela são alguns exemplos de padrões elementares. No Código 2.1, está ilustrada a estrutura do padrão elementar repetição com sentinela. A descrição completa deste padrão elementar pode ser visualizada na Tabela 3.2 localizada na Seção 3.1 do Capítulo 3.

Código Fonte 2.1: Padrão elementar de repetição com sentinela

```
1 Inicializar variável sentinela;
2 Enquanto (variável sentinela satifaz a condição) faça início
3     Processar o elemento;
4     Atualizar a variável sentinela;
5 fim;
```

Em 2000, Proulx [Pro00] publicou um *framework* para um curso de introdução à programação baseado em padrões elementares e padrões de projetos. O objetivo desse trabalho era ajudar o aluno no desenvolvimento da habilidade de raciocinar e projetar soluções. Nesse trabalho, ele descreve padrões para nomear variáveis, constantes e funções; para leitura, processamento e escrita de dados; além de padrões de seleção, repetição e conversão.

### 2.3.2 Padrões Algorítmicos

Os padrões algorítmicos são blocos de códigos focados na semântica do problema. Diferentemente dos padrões elementares, os algorítmicos se referem a uma classificação dos próprios problemas.

O principal objetivo desse tipo de padrão é auxiliar o aluno a aprimorar sua habilidade de resolução de problemas por meio do raciocínio por analogia [Mul05].

Padrões de contagem, acumulação e computação de valores extremos, assim como, busca por itens, checagem de valores e identificação de elementos mais frequentes, são alguns dos exemplos de padrões algorítmicos. No Código-Fonte 2.2, encontra-se ilustrado o padrão de valores extremos.

Código Fonte 2.2: Padrão algorítmico de computação de valores extremos.

```
1 Inicializar maior_Valor com o valor do primeiro elemento da lista;
2     Enquanto (ainda existir elemento na lista) faça
3         valor recebe o valor do próximo elemento da lista;
4         Se o valor é maior do que o maior_valor
5             O maior_valor recebe valor;
```

A *Pattern-Oriented Instruction* (POI) é uma abordagem pedagógica que incorpora os padrões algorítmicos no ensino de alunos iniciantes [Mul05]. O objetivo da POI é desenvolver a habilidade de resolução de problemas de programação [HM08].

Para dar suporte a essa metodologia de ensino, Ginat *et. al* publicou o livro *Patterns in computer science* [GHC+01], apenas editado em Hebraíco, contendo 30 (trinta) padrões algorítmicos. Segundo os autores, esses padrões abrangem os principais tipos de problemas apresentados em um curso introdutório de ciência da computação e tratam dos problemas mais básicos aos mais difíceis.

### 2.3.3 Discussão

Os padrões para ensino de programação são importantes ferramentas. Entretanto, poucos são os cursos que atualmente adotam abordagens de ensino focadas na utilização de tais padrões.

Contudo, é importante ressaltar que o simples uso dos padrões em sala de aula não é suficiente para tornar o aluno um bom resolvedor de problemas de programação. O aluno iniciante, mesmo conhecendo os padrões, pode não conseguir selecionar qual deverá ser utilizado para resolver o problema, ou ainda, não saber como combinar os padrões para chegar à solução. Portanto, torna-se útil disponibilizar ao aluno ambientes computacionais que apresentem a documentação dos padrões quando necessário, que auxiliem o aluno na criação do esqueleto do padrão no momento da codificação e que o ajude a identificar problemas similares a partir dos padrões utilizados. Com isso, é possível aprimorar a habilidade do aluno em resolver problemas de programação por meio do raciocínio por analogia.

Os padrões mencionados anteriormente, apesar das semelhanças, possuem características distintas. Desse modo, acredita-se que é importante o aluno iniciar o aprendizado com padrões elementares a fim de dominar os elementos básicos da linguagem de programação estudada. Em seguida, o ensino deve prosseguir nos estudos por meio dos padrões algorítmicos, nos quais o aluno irá lidar com soluções de problemas mais complexos.

Mais informações, sobre a utilização de padrões no ensino de alunos iniciantes do curso de introdução à programação, são encontradas em [BKP+99; BBD+01; GHC+01; Pro00; MHA04; BDM04; MH05; dBADM05; Mul05; Del05; BD06; MGH07; HM08; Mul08]

## 2.4 Agentes Inteligentes de Diálogo

Durante anos, o homem vem tentando criar servos perfeitos, os robôs, capazes de executar tarefas do cotidiano das pessoas ou até ações que ponham a vida dos seres humanos em risco. Nesse sentido, não era suficiente apenas criar apenas uma máquina capaz de realizar tarefas

pré-definidas, mas construir uma máquina que fosse capaz de tomar decisões sozinha, para solucionar o problema. O matemático Alan Turing lançou em um dos seus trabalhos uma questão simples que até hoje muitos pesquisadores tentam responder. "Podem as máquinas pensar?"[Leo05].

Essa questão deu origem ao Jogo de Imitação, também conhecido por Teste de Turing. O Jogo de Imitação é constituído por 3 (três) personagens: um interrogador (I), um homem (H) e uma mulher (M). Os personagens H e M estão escondidos do interrogador em salas separadas. O objetivo do jogo é fazer o interrogador identificar em qual sala está o homem e a mulher, mediante um conjunto de perguntas efetuadas.

A complexidade desse jogo está na possibilidade dos personagens H e M fornecerem informações que possam confundir o interrogador como, por exemplo, se perguntado pelo cabelo, o homem pode informar que possui cabelos longos.

O Teste de Turing consiste em substituir um dos personagens, homem ou mulher, por uma máquina capaz de dialogar com o interrogador. Nesse cenário, seria possível fazer com que o interrogador não identificasse quem era o homem e quem era a máquina? Baseado nesse cenário, muitas pesquisas sobre agentes inteligentes de diálogo, popularmente conhecidos por *chatterbots* ou simplesmente *bots*, foram realizadas ao longo dos anos.

Um agente inteligente de diálogo é um programa capaz de simular uma conversação com um ser humano, com o objetivo de fazer o interlocutor pensar que está falando com outro humano [LNT03]. Essa busca pela compreensão e simulação do comportamento humano é um dos alvos de pesquisa da Inteligência Artificial.

Com o advento da *Internet* e os avanços da *Web*, é possível encontrar o emprego dos *chatterbots* nos mais diferentes contextos: em *sites* comerciais, exercendo o papel de um recepcionista e até mesmo de recomendador de produtos; em *sites* de entretenimento, como personagens de jogos virtuais; em *sites* de ajuda ao consumidor, auxiliando na resolução de problemas em produtos; na educação a distância, acompanhando o aluno no seu aprendizado; dentre outros [LCDT03].

### 2.4.1 Estratégias de conversação

Em se tratando de conversação, existem 2 (dois) tópicos a serem tratados [Leo05]:

- O algoritmo utilizado na obtenção da resposta para uma determinada pergunta;

- As estratégias de condução do diálogo, ou seja, a maneira na qual a conversa é iniciada, direcionada e mantida.

A escolha de uma resposta pode ser realizada de diferentes formas, sendo que cada uma delas apresenta diferentes características como, por exemplo, facilidades de representação da informação, precisão da informação recuperada, rapidez na recuperação da informação, dentre outras [Leo05]. Nesse trabalho, destacam-se 2 (duas) estratégias, sendo essas:

- **Casamento de padrões:** um conjunto de palavras-chave, organizadas com prioridade, são utilizadas para indexar as respostas. Uma vez que a palavra é identificada na base, sua resposta é selecionada;
- **Raciocínio baseado em casos:** a resposta é dada mediante um conjunto de casos armazenados na memória. É identificado na memória o caso que possui a pergunta mais similar à efetuada e sua resposta é utilizada, considerando as devidas adaptações.

Considerando o segundo tópico, pode-se apontar um conjunto de estratégias de condução do diálogo, visando fornecer ao usuário uma ilusão de inteligência e fluência do *chatterbot*, sendo essas [Leo05]:

- Manter a iniciativa no diálogo por meio do constante questionamento;
- Formular respostas com partes da pergunta do usuário;
- Realizar perguntas que aprofundem o diálogo;
- Permanecer no mesmo tópico, questionando o interlocutor quando este mudar de assunto;
- Mudar o tema da conversa quando esta se tornar repetitiva;
- Fazer comentários controversos ou humorísticos relacionado ao tema da conversa.

Essas estratégias de conversação estão presentes nos principais agentes inteligentes de diálogo para que a conversa entre o *bot* e o humano ocorra o mais natural possível. Hoje, um dos principais *chatterbots* que apresenta algumas dessas características é o ALICE, que será detalhado a seguir.

### 2.4.2 ALICE

Alice é um dos *chatterbots*, atualmente, mais populares [Leo05]. Foi desenvolvido em 1995 por Richard Wallace na *Lehigh University*. As características que o tornam um dos precursores de vários outros *chatbots* é o alto poder de conversação e sua interface gráfica que estimula o diálogo.

A base de dados de Alice é bastante rica em fatos, citações e idéias do seu criador. Apresenta um vocabulário com mais de 5000 palavras. Esse *bot* foi desenvolvido para fazer com que o interlocutor se sinta confortável no diálogo e, por isso, Alice é capaz de conversar sobre vários assuntos, contar piadas e até cantar [LNT03].

Basicamente, a arquitetura de um *chatterbot* é constituída por uma interface gráfica que interage diretamente com o interlocutor; uma máquina de inferência, que analisa a pergunta realizada pelo interlocutor e infere uma resposta a partir de uma base de conhecimento; a base de conhecimento é o cérebro do *bot*.

A base de conhecimento e o comportamento de Alice foram construídos e implementados utilizando a linguagem AIML [PCR08]. A AIML, que será vista com mais detalhes na Seção 2.4.3, é uma linguagem de marcação derivada da linguagem XML.

### 2.4.3 AIML

Essa linguagem permite a definição de estímulos – *patterns* – e a correspondente resposta do sistema – *template*. No Código 2.3 está ilustrada a representação de um simples conhecimento em AIML.

Código Fonte 2.3: Exemplo de um código AIML.

```
1 <category>
2 <pattern> pergunta </pattern>
3 <template> resposta </template>
4 </category>
```

A AIML é constituída por mais de 20 (vinte) *tags* que possibilitam a criação desde diálogo simples, a exemplo do apresentado no Código 2.3, a diálogos invocando respostas passadas, escolhendo aleatoriamente uma resposta, entre várias, para uma mesma pergunta e responder invocando trechos do enunciado da pergunta feito pelo interlocutor. A documentação completa sobre tais *tags* pode ser acessada em (http://www.alicebot.org/documentation/), sendo que as principais são:

- **<aiml>:** inicia e termina um bloco AIML;
- **<category>:** identifica uma unidade de conhecimento na base de conhecimento;
- **<pattern>:** identifica um estímulo;
- **<template>:** contém a resposta para o estímulo;
- **<that>:** armazena a declaração anterior do *chatterbot*, possibilitando respostas dentro de um contexto;
- **<random>:** escolhe uma resposta randomicamente, dentre as cadastradas na base.

A linguagem AIML é simples e fácil de usar. Entretanto, em algumas situações, a aquisição e representação do conhecimento geram algumas dificuldades, visto que para um diálogo poderoso é necessário uma base de conhecimento considerável sobre o assunto.

### 2.4.4 Exemplos de agentes inteligentes de diálogo aplicados na educação

Durante anos, vários *chatterbots* foram desenvolvidos. A seguir, serão apresentados alguns desses *bots* criados com o intuito de auxiliar o ensino/aprendizagem do aluno.

#### Doroty

Doroty [LT06; Leo05; LTV+07] é uma adaptação do *chatterbot* ALICE que permite o treinamento de gerente de redes pouco experientes. Esse *chatterbot* é capaz de monitorar uma rede de computadores por meio do protocolo SNMP.

Quando um usuário realiza um pergunta em linguagem natural para Doroty, este processa a informação e devolve da sua base de conhecimento, implementada na linguagem AIML, a resposta teórica para o questionamento. Caso sejam necessárias informações práticas da rede, Doroty por meio do módulo central ativa o módulo coletor que fornece tais informações a partir do histórico existente na base de dados e de consultas à rede por meio do protocolo SNMP. A arquitetura que possibilita o funcionamento do *chatterbot* Doroty está ilustrada na Figura 2.3.

Doroty foi implementado em Java, sua base de conhecimento desenvolvida a partir de arquivos AIML e seu histórico de informações da rede foi armazenado no banco de dados

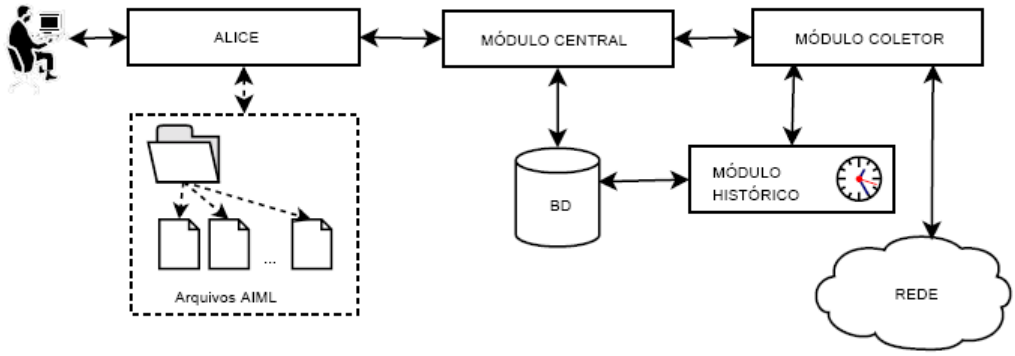



Figura 2.3: Arquitetura do *chatterbot* Doroty.

MySQL. A *interface* com o usuário é simples e objetiva. Esta *interface* é constituida por um texto de boas-vindas, uma caixa de entrada para enviar a pergunta ao *chatterbot*, uma caixa de saída, pela qual Doroty responde o questionamento efetuado, e uma região de últimas notícias, em que são apresentadas novidades sobre Doroty. Na Figura 2.4, está ilustrada a interface de Doroty.

### Meara

Meara [LNT03] é um *chatterbot* para auxiliar o aprendizado de redes de computadores por estudantes de cursos técnicos e de graduação.

Quando um aluno questiona sobre um assunto de redes de computadores, Meara busca em sua base de conhecimento a resposta para a pergunta. Caso não exista na base informações suficientes para responder o aluno, Meara devolve a seguinte mensagem: "Não conheço nada sobre <assunto>. Você gostaria que eu procurasse informações na *internet* para você?". Se o aluno concordar com a busca da informação, o *chatterbot* ativa seu módulo de busca na *net* e devolve uma lista de páginas com informações relevantes sobre o assunto.

### Elektra

Elektra [LCDT03] é uma professora virtual capaz de responder sobre questionamentos de Física, voltada para alunos do Ensino Médio, que estão se preparando para se submeter ao concurso de vestibular.

Esse *chatterbot* foi desenvolvido baseando-se na arquitetura do ALICE e seu conhecimento está armazenado em arquivos no formato AIML. Uma das principais características de Elektra é a capacidade de responder de diferentes formas uma mesma pergunta, o que


Figura 2.4: Interface do *chatterbot* Doroty.

torna o diálogo mais real e motivador.

### 2.4.5 Discussão

O agente inteligente de diálogo é bastante popular e os avanços da *Internet* possibilitaram o surgimento de inúmeras aplicações dessa tecnologia. Hoje, os *chatterbots* são empregados em diferentes áreas como: comércio eletrônico, atendimento virtual de clientes e educação à distância.

O emprego dos *bots* na educação à distância talvez seja uma das áreas mais promissoras, uma vez que esse agente está disponível a todo instante para tirar dúvidas dos alunos, mesmo quando o aluno possui horário distinto dos professores e colegas de classe.

Além disso, é possível utilizar os *chatbots* como forma de interação homem-máquina para apoiar outras tecnologias da Inteligência Artificial, um exemplo é o *chatterbot* para área imobiliária, criado por Krauss [KF07], que juntamente com um sistema de raciocínio baseado em casos recomenda imóveis aos seus clientes.

Hoje, é possível encontrar facilmente interpretadores para desenvolvimento de *chatterbots* utilizando a linguagem AIML. Porém, vale ressaltar que é necessário esforço para adquirir e implementar a base de conhecimento nessa linguagem, a depender do nível de conversação que se deseja fornecer ao *bot*.

No ensino de programação para iniciantes, um agente inteligente de diálogo pode auxiliar o aluno debatendo sobre os aspectos inerentes da linguagem, os padrões de programação aprendidos em sala de aula e até discutir sobre como resolver um determinado problema.

# Capítulo 3

# Trabalhos Relacionados

Esse capítulo destina-se à apresentação dos trabalhos relacionados aos temas fundamentados no Capítulo 2. E, tem como principal objetivo analisar criticamente as principais ferramentas capazes de auxiliar o aluno iniciante no desenvolvimento do raciocínio por meio de analogia para resolução de problemas de programação.

Na Seção 3.1 será analisado o ProPAT *plugin* proposto por Delgado et al [BDM04]. Na Seção 3.2, será apresentado o sistema de ensino de lógica de programação – SELP – desenvolvido por Oliveira [OGP08]. Na Seção 3.3, será descrita a abordagem pedagógica POI, criada por Muller et al [MHA04]. Na Seção 3.4 será apresentado um ambiente de aprendizagem de lógica de programação baseado em casos criado por Koslosky [Kos99]. Na Seção 3.5, será analisada uma ferramenta de construção de abstrações em lógica de programação desenvolvida por Mattos [Mat00]. Por fim, na Seção 3.6, será realizada uma análise comparativa dessas ferramentas com o sistema proposto nesta dissertação.

## 3.1 ProPAT

O ProPAT [BDM04; dBADM05; Del05; BD06] é um *plugin* para o Eclipse IDE, que auxilia os alunos do cursos de introdução à programação no aprendizado dessa disciplina. Tal *plugin* possibilita que os alunos iniciantes aprendam a programar por meio de padrões.

O ProPAT *plugin* é constituído por 2 (duas) perspectivas distintas: a do aluno e a do professor. Na perspectiva do aluno, é possível escolher os exercícios de programação, disponibilizados pelo professor, e construir as soluções destes a partir do "esqueleto"de padrões elementares. Já na perspectiva do professor, é possível inserir e manter os exercícios e os padrões. Nas Figuras 3.1 e 3.2, estão ilustradas as perspectivas do aluno e do professor,

respectivamente.

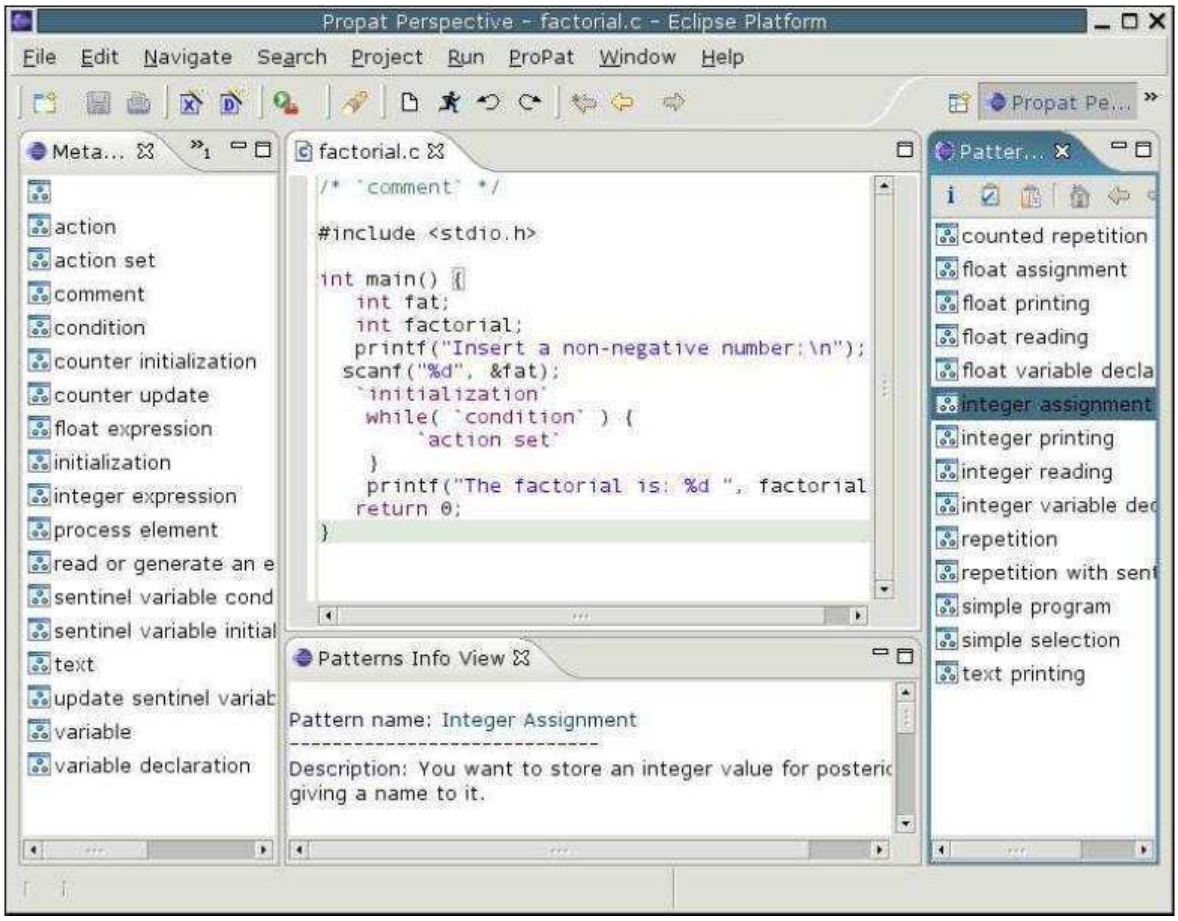



Figura 3.1: Perspectiva do Aluno no ProPAT. Adaptado de [Del05].

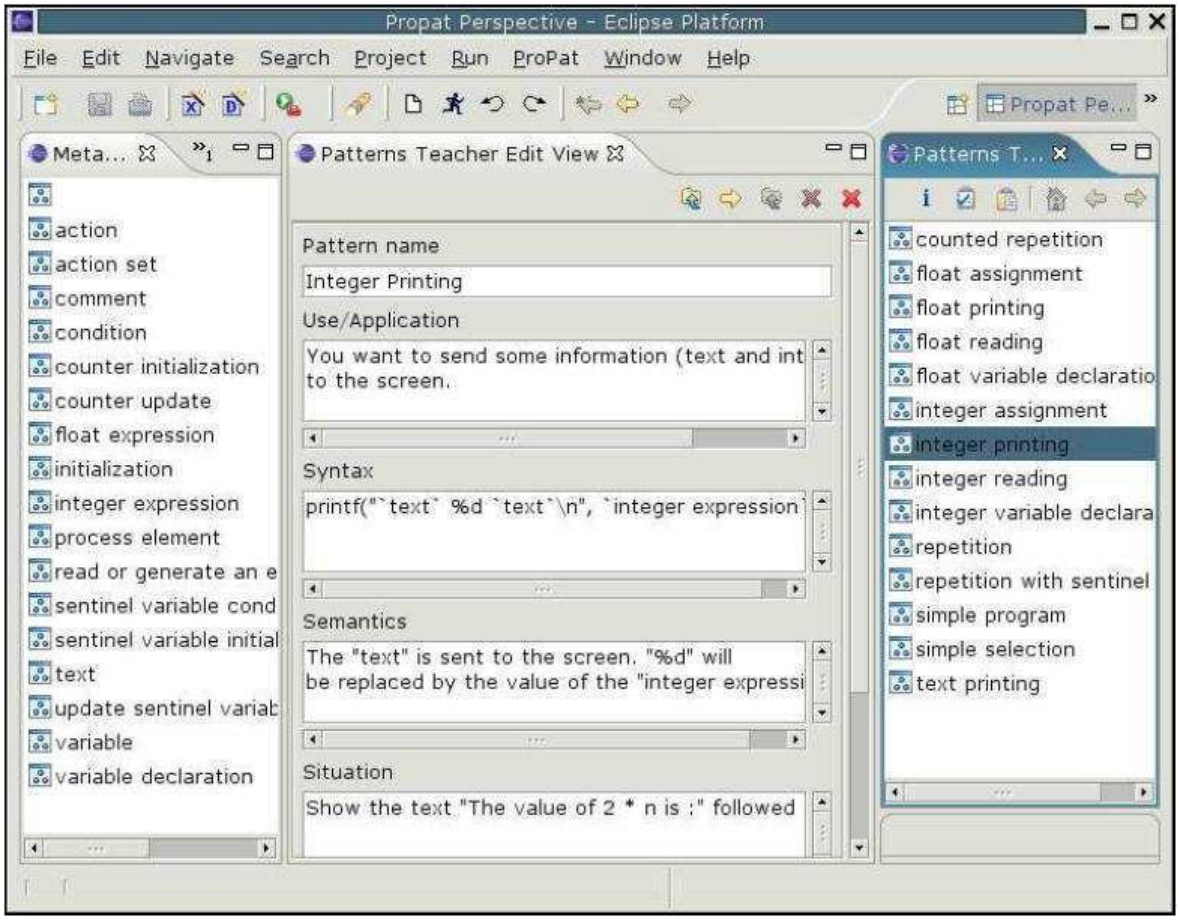



Figura 3.2: Perspectiva do Professor no ProPAT. Adaptado de [Del05].

Para manutenção dos padrões é necessário que o professor siga o *template* definido por Delgado [Del05] e descrito na Tabela 3.1. Para exemplificar, foi utilizado o padrão Repetição por Sentinela, exibido no Código 3.1 e descrito na Tabela 3.2, seguindo o *template* definido por Delgado.

Tabela 3.1: *Template* da descrição de padrões elementares no ProPAT. Adaptado de [Del05]

| **Elemento** | **Descrição** |
|---|---|
| Nome do Padrão | Nome do padrão elementar, geralmente, indicado pela estrutura de controle. |
| Intenção do padrão | O que o padrão faz? |
| Intenção pedagógica | Qual a intenção do professor com o padrão. |
| Nomes Alternativos | Outro nomes conhecidos do padrão. |
| Motivação | Um cenário que ilustra um problema e como este é resolvido pelo padrão. |
| Estrutura da Solução 1: Sintaxe | Pseudo-código do padrão. |
| Estrutura da Solução 2: Semântica | Descreve como o padrão trabalha por meio de uma descrição textual ou um diagrama de fluxo. |
| Aplicabilidade | Em quais situações o padrão pode ser aplicado? |
| Pré-requisitos | Conceitos que o aluno já deve conhecer. |
| Consequências | Quais os resultados do emprego do padrão? |
| Implementação | Sugestões de implementação do padrão que devem ser conhecidas pelo aluno. Por exemplo, definição de tipos de dados, evitar a divisão por zero, entre outras. |
| Código de Exemplo | Fragmentos de código da implementação do padrão em uma linguagem. |
| Usos Conhecidos | Exemplos do uso do padrão em exercícios já resolvidos pelo aluno. |
| Padrões Relacionados | Quais padrões estão relacionados a este? Quais suas diferenças? |

Código Fonte 3.1: Sintaxe do padrão repetição por sentinela

```
1 <Inicializações>
2 <Inicialização da variável sentinela>
3 while (<condição da variável sentinela>) {
4         <Leitura/Geração de um elemento da sequência>
5         <Processar elemento>
6         <Atualização da variável sentinela>
7 }
```

Código Fonte 3.2: Exemplo do padrão repetição com sentinela

```
1 soma = 0;
2 printf("Digite o número inteiro: ");
3 scanf("%d", &numero); /* leitura do primeiro número */
4 while (numero != 0) {
5         soma = soma + numero;
6         printf("Digite o número inteiro: ");
7         scanf("%d", &numero); /* leitura do número seguinte */
8 }
```

Tabela 3.2: Descrição do padrão Repetição por Sentinela. Adaptado de [Del05].

| **Elemento** | **Descrição** |
|---|---|
| Nome do Padrão | Repetição com Sentinela. |
| Intenção do padrão | Processar um sequência com um número desconhecido de elementos e terminada com um valor dado. |
| Intenção pedagógica | Fazer com que o aluno aprenda: (i) uma forma de processar uma sequência sem uso de contador de elementos; (ii) usar um valor constante como indicador de fim de sequência. |
| Nomes Alternativos | |
| Motivação | Processar uma sequência de elementos que são números. Os elementos podem ser lidos (entradas do programa) ou gerados. Processar uma sequência pode ser: (i) executar ações na sequência, por exemplo, contar seus elementos, somar ou modificá-los; (ii) verificar algumas propriedades sobre a própria sequência ou subsequência; (iii) verificar propriedades sobre cada elemento ou entre elementos. |
| Estrutura da Solução 1: Sintaxe | Pseudo-código do padrão está descrito no Código-Fonte 3.1. |
| Estrutura da Solução 2: Semântica | A inicialização de variáveis e inicialização da sentinela para condição verdadeira são executadas uma única vez. Em seguida, a condição da sentinela é verificada: (i) um elemento da sequência é lido/gerado; (ii) processado e, eventualmente, (iii) a variável sentinela é atualizada. Em seguida, a condição da variável sentinela se torna falsa. |
| Aplicabilidade | Você quer repetir um conjunto de ações que, em geral, está relacionado ao processamento de uma sequência de elementos. Os elementos da sequência podem ser lidos ou gerados. A quantidade de elementos é desconhecida, mas o fim da sequência é indicado por um valor sentinela. Isso quer dizer que o número de repetições depende da variável sentinela e, portanto, essa variável deve estar na condição de repetição. |
| Pré-requisitos | Declaração de variável inteira, conjunto de ações, inicialização e leitura. |
| Consequências | |
| Implementação | A atualização e a condição da variável sentinela devem ser feitas corretamente, caso contrário, o programa pode entrar em um "laço infinito". |
| Código de Exemplo | Faça um programa que leia uma sequência de números inteiros terminados por zero e calcule sua soma. A implementação desse problema é encontrado no Código 3.2. |
| Usos Conhecidos | Exemplos do uso do padrão em exercícios já resolvidos pelo aluno |
| Padrões Relacionados | Repetição contatada e Repetição com indicador de passagem. |

Outra característica relevante no ProPAT é o módulo de depuração automática denominado ProPAT _ DEBUG. Este módulo é capaz de diagnosticar falhas semânticas e lógicas na solução algorítmica do aluno, gerando hipóteses dessas falhas e se comunicando com o aluno por meio de mensagens baseadas nos Padrões Elementares.

Embora o ProPAT auxilie o aluno no aprendizado de resoluções de problemas de programação, por meio de raciocínio por analogia, utilizando padrões elementares, esse *plugin* não apresenta mecanismos capazes de fazer o aluno refletir sobre as similaridades entre os problemas ou de recomendar os problemas resolvidos anteriormente, semelhantes ao atual.

Além disso, acredita-se que o Eclipse IDE pode proporcionar aos alunos iniciantes dificuldades extras no momento do aprendizado, visto que é uma ferramenta profissional com inúmeras funcionalidades, podendo desconcentrá-lo e até confudi-lo.

## 3.2 SELP

O Sistema para Ensino de Lógica de Programação (SELP) [OGP08] é uma ferramenta para assistência à aprendizagem de programação baseada em padrões pedagógicos. Tal ferramenta atua como um assistente, guiando o aluno na resolução do problema, caso seja necessário.

Essa ferramenta é constituída por 3 (três) módulos, sendo esses [OGP08]:

- **Módulo de Cadastro de *Template*:** módulo para cadastro dos padrões de programação. Na Figura 3.3, está ilustrada a tela de cadastro do *template*;

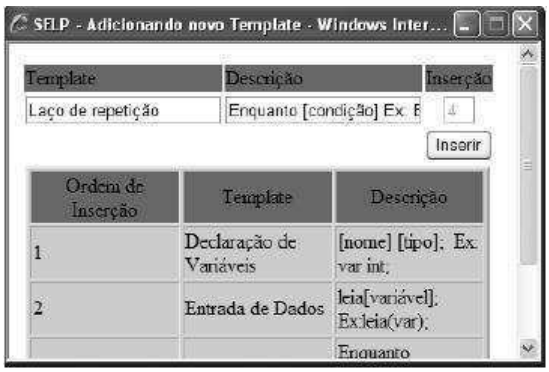



Figura 3.3: Módulo de cadastro de *template*

- **Módulo de Cadastro de Enunciado:** módulo que permite ao professor cadastrar os problemas a serem solucionados pelos alunos. Na Figura 3.4, está ilustrado esse módulo;

- **Módulo de Resolução:** possibilita a resolução do problema pelo aluno. O aluno pode ser guiado na solução ou pode desenvolvê-la livremente. Ressalta-se que na resolução guiada, é exibido ao aluno iniciante o padrão sugerido pelo professor para resolução do problema. Nas Figuras 3.5(a) e 3.5(b), estão ilustradas as telas de resolução guiada e livre, respectivamente.

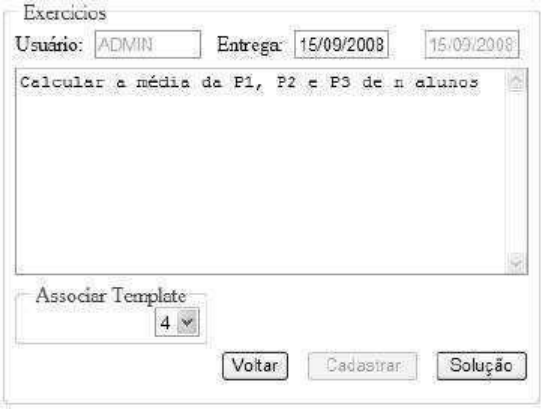
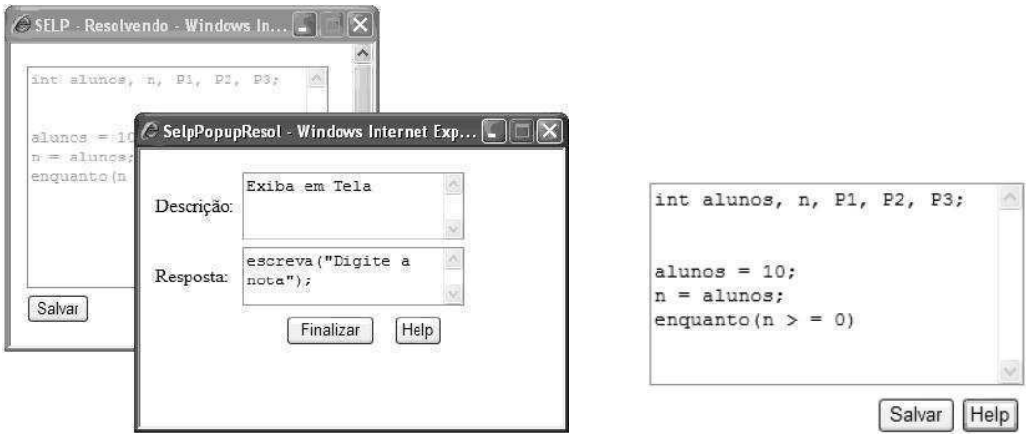



Figura 3.4: Módulo de cadastro do enunciado do problema.




(a) Resolução guiada. (b) Resolução livre.

Figura 3.5: Módulo de resolução de problemas.

O SELP possibilita a utilização dos padrões pelos alunos na resolução dos problemas, com uma interface simples e *Web*. Entretanto, diferente do ProPAT, o SELP não segue um *template* para documentação dos padrões, como o proposto por Delgado [Del05] .

Embora o SELP guie o aluno na resolução dos problemas, ele não se preocupa em fazer o aluno refletir sobre as similaridades entre os problemas, o que poderia ajudar o aluno no desenvolvimento da capacidade de resolver problemas por meio de analogia.

## 3.3 *Pattern-Oriented Instruction*

O *Pattern-Oriented Instruction* (POI) [MHA04] é uma abordagem pedagógica que incorpora padrões algorítmicos no ensino introdutório de programação. O *Computer Societ Group* da Universidade de *Tel-Aviv* publicou inúmeros trabalhos que fundamentam e justificam a uti-

lização dessa metodologia no ensino [GHC+01; MHA04; Mul05; MH05; MGH07; Mul08; HM08].

Ao longo dos trabalhos, o grupo publicou um livro contendo 30 (trinta) padrões algorítmicos que, segundo os autores, cobrem os principais problemas de programação vistos em um curso introdutório [GHC+01]. Em seguida, apresentou à comunidade um artigo que relata a importância do ensino por meio dessa metodologia, juntamente com guia para sua aplicação [MHA04].

A partir dessas publicações, foram realizados estudos sobre a importância do ensino baseado em padrões para o desenvolvimento da habilidade do raciocínio por meio de analogia [Mul05], como também, para ampliação da aptidão de decompor problemas na resolução de problemas de programação [MGH07] e, ainda, para o aprimoramento da habilidade de abstração de soluções [HM08].

A metodologia proposta pelo grupo possui forte embasamento teórico e prático, visto que está fundamentada em anos de pesquisa sobre o ensino de programação por meio de padrões e em estudos realizados com 300 (trezentos) alunos iniciantes, cujo intuito era comprovar a eficiência do POI.

Embora o grupo tenha disponibilizado esse arcabouço teórico, este não desenvolveu ferramentas computacionais que suportem o emprego da metodologia e auxiliem os professores no ensino presencial e não presencial.

Além disso, também é preciso ressaltar que o livro contendo os padrões foi publicado apenas em Hebraico e não existe, até o momento, traduções em outros idiomas, embora, em seus artigos eles mencionam alguns dos padrões desenvolvidos.

## 3.4 Aprendizagem Baseada em Casos - Um Ambiente de Ensino de Lógica de Programação

Koslosky [Kos99] desenvolveu o protótipo de um ambiente de suporte de ensino/aprendizagem de lógica de programação, por meio do auxílio de um sistema de raciocínio baseado em casos.

Nesse trabalho de Koslosky, o problema no caso é constituído pelo enunciado, enquanto que a solução é composta por: algoritmo da solução, quantidade de instruções (QI), quantidade de variáveis (QV), quantidade de estruturas de repetição (QER) e de seleção (QES),

a justificativa do aluno para os elementos utilizados e os comentários do professor sobre a solução.

A similaridade é calculada pelo algoritmo do vizinho mais próximo. Tal cálculo é efetuado a partir do enunciando do problema e os atributos QI, QV, QER e QES, sendo atribuído um peso para cada atributo, indicando sua importância na representação do caso. Na Tabela 3.3 estão definidos os pesos utilizados.

Tabela 3.3: Valores dos pesos dos atributos

| Atributo | Peso |
|---|---|
| Quantidade de instruções (QI) | 2 |
| Quantidade de variáveis (QV) | 1 |
| Quantidade de estruturas de repetição (QER) | 4 |
| Quantidade de estruturas de seleção (QES) | 4 |

Nessa ferramenta utiliza-se a função limiar para o cálculo da similaridade local dos atributos numéricos. Para isso, foram definidas tabelas de similaridade indicando o grau de semelhança entre os atributos de dois casos. O grau de similaridade dos atributos QV, QER e QES é apresentado na Tabela 3.4(a), enquanto o atributo QI está apresentado na Tabela 3.4(b).

Tabela 3.4: Tabelas de similaridade para os atributos numéricos.

(a) Atributos QV, QER e QES

| **Distância** | **Similaridade** |
|---|---|
| 0 | 1 |
| 1 | 0,5 |
| > 1 | 0 |

(b) Atributo QI.

| **Distância** | **Similaridade** |
|---|---|
| 0 | 1 |
| 1 | 0,75 |
| 2 | 0,5 |
| 3 | 0,25 |
| > 3 | 0 |

Após realizar a recuperação dos casos, o ambiente permite que o aluno analise os casos recuperados e os comentários emitidos pelo professor para os mesmos e reutilize esse conhecimento na resolução do novo problema. Assim que a solução é construída pelo aluno, o problema é armazenado no sistema para ser revisado e comentado pelo professor.

Ainda que o ambiente desenvolvido por Koslosky consiga recuperar problemas similares ao que o aluno está solucionando, sua recuperação é bastante ingênua. Isso ocorre porque os índices utilizados na representação do caso são simples e pouco representativos, principalmente quando se trata de problemas complexos.

Para exemplificar, veja os problemas simples a seguir:

**a)** *Faça um programa que leia uma lista de números positivos. A leitura deve ser encerrada quando for informado um número negativo. Em seguida, imprima o maior e o menor valor dessa lista.*

**b)** *Faça um programa que leia duas listas de inteiros. Se as listas forem do mesmo tamanho, imprima a soma dos elementos de índices pares.*

Para solucionar tais problemas, um aluno poderia criar soluções semelhantes aos Códigos-Fonte 3.3 e 3.4, para os problemas a) e b), respectivamente.

Código Fonte 3.3: Solução do problema a)

```
1  leia(valor);
2  maior_valor := valor;
3  menor_valor := valor;
4  Enquanto (valor >= 0) faça
5     Se (valor > maior_valor) então
6        maior_valor := valor;
7     Senão Se (valor < menor_valor) então
8           menor_valor := valor;
9     leia(valor)
10 imprima "Maior valor: " + maior_valor + "Menor Valor: " + menor_valor;
```

Código Fonte 3.4: Solução do problema b)

```
1  leia(vetor1)
2  leia(vetor2)
3  Se (tamanho(vetor1) = tamanho(vetor2)) então
4    imprima "Vetores com mesmo tamanho.";
5    Para i = 0 até tamanho(vetor1) faça
6       Se ((i resto 2) = 0) então
7          imprima(vetor1[i] + vetor2[i]);
8  Senão
9    imprima "Vetores com tamanhos distintos.";
```

De acordo com os dados apresentados na Tabela 3.5, os problemas são extremamente semelhantes, mediante o cálculo de similaridade de Koslosky, pois, exceto a quantidade de instruções (QI), os índices possuem valores idênticos. Tal fato é comprovado com o cálculo da similaridade global entre esses dois problemas, conforme apresentado na Equação 3.1. O resultado dessa equação é 0,954, valor muito próximo de 1, indicando alto grau de similaridade.

$$Sim(a,b) = \frac{QI \times 2 + QV \times 1 + QER \times 4 + QES \times 4}{2 + 1 + 4 + 4} = \frac{10,5}{11} = 0,954. \qquad (3.1)$$

Tabela 3.5: Valores dos pesos dos atributos

| **Atributo** | **Problema a)** | **Problema b)** | **Sim. Local** | **Peso** |
|---|---|---|---|---|
| Quantidade de instruções (QI) | 10 | 9 | 0,75 | 2 |
| Quantidade de variáveis (QV) | 3 | 3 | 1 | 1 |
| Quantidade de estruturas de repetição (QER) | 1 | 1 | 1 | 4 |
| Quantidade de estruturas de seleção (QES) | 2 | 2 | 1 | 4 |

Entretanto, é fácil observar que as soluções são completamente distintas. Isso ocorre porque o principal determinante para a semelhança entre dois algoritmos é a forma pelo qual as estruturas – de seleção ou de repetição – estão aninhadas ou sequenciadas e não apenas pela quantidade de estruturas.

Destaca-se ainda que, nesse ambiente, não há mecanismo que proporcione ao aluno uma reflexão sobre os aspectos de similaridade entre os problemas, tampouco elementos que possibilitem a utilização de padrões de programação na construção das soluções. Muito embora tais padrões sejam fortemente indicados na resolução de problemas de programação por analogia.

## 3.5 Ferramenta de Construção de Abstrações em Lógica de Programação

Mattos [Mat00; Mat02] apresentou uma ferramenta de auxílio ao aprendizado de lógica de programação, assistido por um sistema de raciocínio baseado em casos.

A pesquisa se iniciou com o desenvolvimento de um sistema especialista baseado em regras [MFL99], cuja função era auxiliar o aluno na compreensão do problema mediante o seu enunciado. Esse sistema se baseou em uma metodologia, por ele criada, contendo 8 (oito) passos para resolução do problema de programação, executados a partir das questões a seguir.

1. **Quais as "variáveis"conhecidas?** Quais as informações que podem ser obtidas a partir do enunciado do problema;

2. **O que precisa ser calculado ou executado?** Identificação do escopo da aplicação;

3. **Quais as "variáveis"desconhecidas?** Variáveis para as quais não há informação no enunciado do problema;

4. **O que precisa ser informado pelo usuário?** O que o usuário da aplicação precisa informar para que o sistema realize suas operações;

5. **O que precisa ser impresso para o usuário?** O que a aplicação deve apresentar como resultado;

6. **Realizar um esboço da solução.** Identificar em termos de macro passos, qual a estratégia a ser adotada para solução do problema;

7. **Construir um fluxograma.** Elaborar um fluxograma, utilizando a notação gráfica para representar a lógica da solução;

8. **Construir o teste-de-mesa.** Exercitar as variáveis a partir da execução dos comandos na sequência em que aparecem no gráfico.

Na Figura 3.6 está ilustrada uma das telas do sistema especialista desenvolvido em CLIPS 6.0, que tinha como objetivo fazer o aluno alcançar a solução por meio da resolução dessas questões.

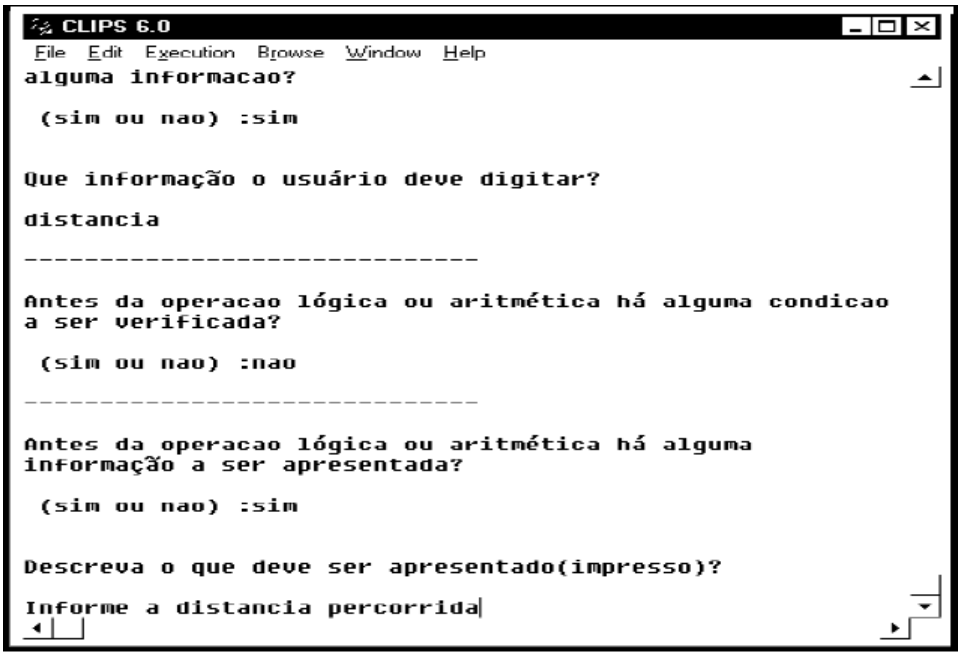



Figura 3.6: Interface do protótipo em Clips 6.0 do sistema especialista.

Com a experiência adquirida na utilização desse protótipo, foi desenvolvido um sistema de raciocínio baseado em casos para auxiliar o aluno a construir a solução do problema. Nas

Figuras 3.7 e 3.8 estão ilustradas a arquitetura geral do sistema e a interface do protótipo, respectivamente.

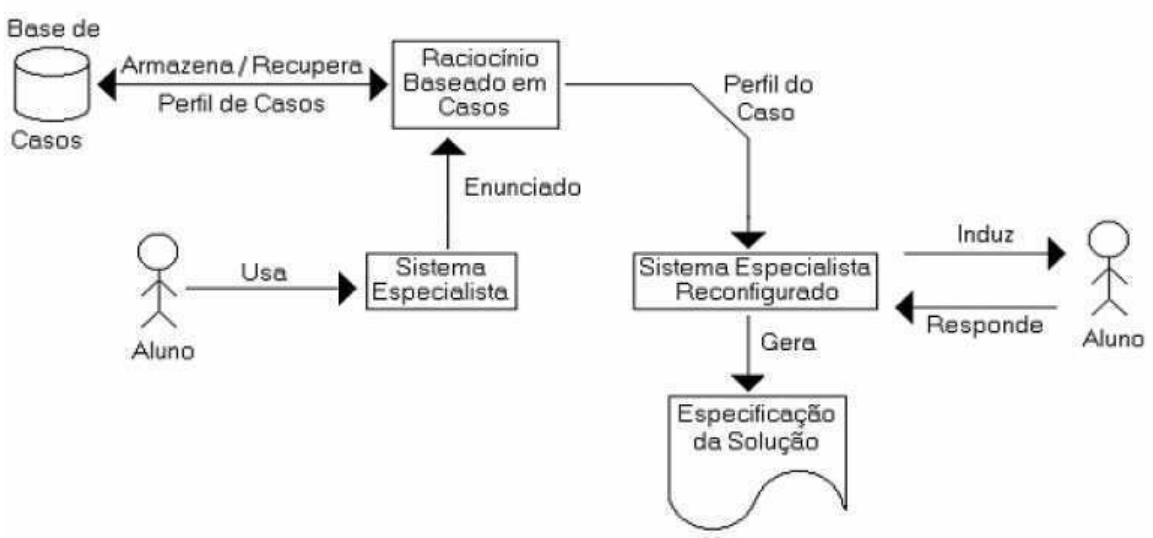



Figura 3.7: Arquitetura geral do sistema.

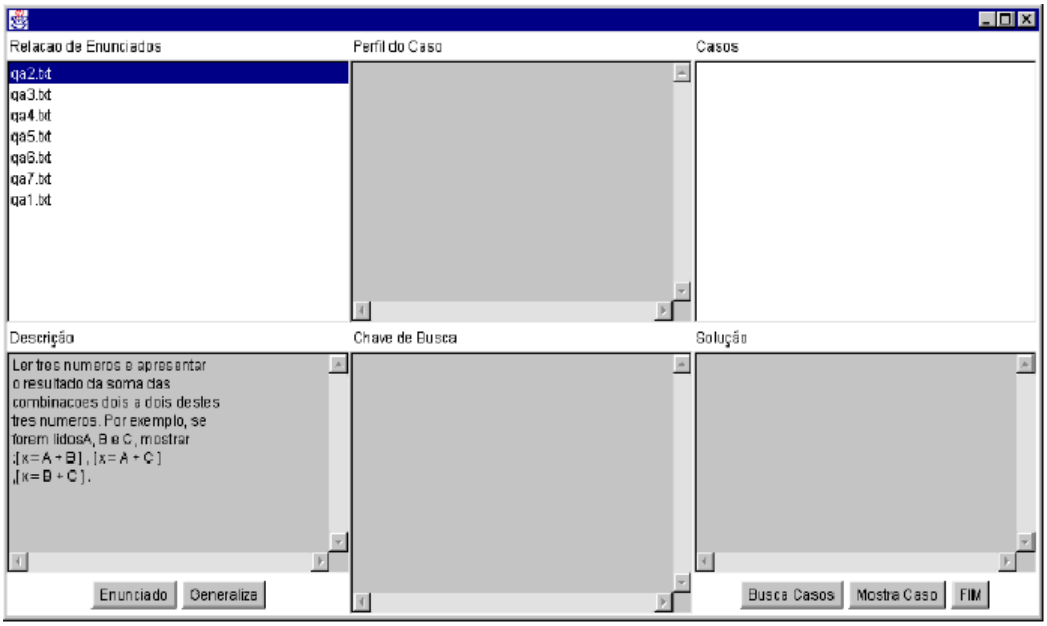



Figura 3.8: Interface do protótipo em Java do RBC.

No RBC desenvolvido por Mattos, o caso é representado por: enunciado, conjunto de palavras-chave extraídas do enunciado do problema, número de passos, número de variáveis, número de constantes e solução do problema. Para esse sistema, as soluções são dadas por uma sequência detalhada de passos necessários para solucionar o problema, de acordo com a metodologia desenvolvida.

A recuperação dos casos ocorre, inicialmente, pelas palavras-chave do enunciado do problema. À medida que o problema é solucionado pelo aluno, a consulta do RBC vai sendo refinada. Assim que o melhor caso é identificado, as informações deste são utilizadas para guiar o sistema especialista nas questões a serem apresentadas ao aluno.

O protótipo do RBC desenvolvido por Mattos ainda é bastante imaturo, de acordo com o próprio autor, uma vez que o trabalho encontra-se na fase de implementação e o sistema não possui estrutura para suportar uma base de casos com grande volume.

Além disso, atributos como número de passos, número de variáveis e constantes para solucionar o problema podem ser considerados pouco significativos com o aumento da base de casos. Essa ferramenta apresenta a mesma falha apontada no ambiente de Koslosky (vide 3.4, problemas *a* e *b*). Ou seja, essa ferramenta, assim como o ambiente de Koslosky, não apresenta mecanismos que auxilie o aluno a refletir sobre as similaridades entre os problemas, tampouco possibilita o ensino de padrões para resolução de problemas de programação por meio de analogias.

Por fim, vale ressaltar que o autor não apresenta nos seus trabalhos um estudo de caso mais detalhado, com dados estatísticos, sobre a utilização do protótipo.

## 3.6 Análise Comparativa

Para realizar a análise comparativa entre os trabalhos relacionados e o proposto nesta dissertação, foram definidas 6 (seis) características: *a)* ambiente de resolução de problemas de programação, *b)* prática do ensino com padrões de programação para alunos iniciantes, *c)* indicação de problemas similares, *d)* auxílio na reflexão sobre as similaridades entre os problemas, *e)* utilização de agentes inteligentes de diálogo e *f)* suporte à depuração do programa do aluno. Cada uma das características será detalhada nessa seção. Por fim, na Tabela 3.6 é apresentada uma sumarização dessa análise.

### 3.6.1 *Característica a)* Ambiente de resolução de problemas de programação

O ProPAT 3.1, o SELP 3.2, o trabalho de Koslosky 3.4 e o de Mattos 3.5 são ambientes que auxiliam o estudante na resolução de problemas de programação como, por exemplo, visualização do enunciado do problema, área de edição para codificação da solução, execução da

solução e histórico com os problemas já solucionados pelo aluno.

### 3.6.2 *Característica b)* Prática do ensino com padrões de programação para alunos iniciantes

A comunidade de educação em computação incentiva a utilização de padrões no ensino de programação para alunos iniciantes, como uma forma de fortalecer sua habilidade de resolução de problemas de programação por meio do raciocínio por analogia.

Atualmente, existem algumas metodologias de ensino que seguem essa linha, como é o caso da POI 3.3 e, ferramentas como o ProPAT 3.1 e o SELP 3.2 desenvolvidas para a prática destas metodologias.

De modo geral, essas ferramentas permitem que os professores cadastrem novos padrões e que os alunos os visualizem e os instanciem para resolver novos problemas.

### 3.6.3 *Característica c)* Indicação de problemas similares

O aluno, ao solucionar um problema de programação por meio de analogia, deve identificar quais aspectos de similaridade serão julgados para avaliar o nível de similaridade entre os problemas. Em seguida, avaliar o grau de semelhanças entre os problemas. E, por fim, reutilizar o conhecimento das soluções dos problemas similares na resolução do novo problema.

Alguns ambientes de resolução de problemas de programação auxiliam o aluno na identificação dos problemas similares. Para isso, essas ferramentas utilizam um – ambiente – sistema de raciocínio baseado em casos para recuperar e indicar ao aluno quais os problemas de programação resolvidos similares ao que ele está resolvendo no momento.

Os trabalhos de Koslosky 3.4 e de Mattos 3.5 apresentam esse mecanismo de auxílio ao aluno.

### 3.6.4 *Característica d)* Auxílio na reflexão sobre as similaridades entre os problemas

A partir dos problemas indicados pelo sistema de raciocínio baseado em casos, é possível que o ambiente interaja com o aluno, auxiliando-o na reflexão sobre os aspectos que levaram tais problemas a serem similares.

Uma estratégia, nesse sentido, consiste em o ambiente emitir um julgamento sobre a

similaridade entre os dois problemas avaliados e questionar ao aluno se ele concorda com o julgamento da ferramenta.

### 3.6.5 *Característica e)* Utilização de agentes inteligentes de diálogo

Atualmente, é comum o emprego de agentes inteligentes de diálogo em ambientes virtuais de ensino para interagir com o aluno e sanar dúvidas sobre o conteúdo visto em sala de aula ou conversar sobre assuntos diversos.

Nesse trabalho, o objetivo do agente é simular um professor virtual, capaz de conversar sobre a utilização de padrões de programação para alunos iniciantes, sobre a sintaxe da linguagem de programação escolhida e sobre os aspectos de similaridades entre os problemas.

### 3.6.6 *Característica e)* Suporte à depuração do programa do aluno

Alguns ambientes como o ProPAT adicionam em sua arquitetura um módulo de depuração automática do programa, a fim de diagnosticar falhas na lógica de programação empregada pelo aluno na solução do problema.

Nesse trabalho, essa tarefa não foi totalmente automatizada. O diagnóstico das falhas de lógica de programação fica a cargo do professor que analisa e emite comentários sobre as soluções criadas pelos alunos.

Tabela 3.6: Análise comparativa entre a solução proposta e os trabalhos relacionados.

| **Trabalhos Relacionados** | **a)** | **b)** | **c)** | **d)** | **e)** | **f)** |
|---|---|---|---|---|---|---|
| ProPAT (Seção 3.1) | X | X | | | | X |
| SELP (Seção 3.2) | X | X | | | | |
| POI (Seção 3.3) | | X | | | | |
| Mattos (Seção 3.5) | X | | X | | | |
| Koslosky (Seção 3.4) | X | | X | | | |
| Solução Proposta | X | X | X | X | X | X |

**LEGENDA:**
Requisito 01 Ambiente de resolução de problemas de programação (Seção 3.6.1)
Requisito 02 Recuperação de problemas similares (Seção 3.6.3)
Requisito 03 Uso de padrões pedagógicos (Seção 3.6.2)
Requisito 04 Reflexão sobre as similarides entre os problemas (Seção 3.6.5)
Requisito 05 Uso de agentes inteligentes de diálogo (Seção 3.6.5)
Requisito 06 Suporte à depuração do programa do aluno (Seção 3.6.6)

# Capítulo 4

# *Analogus*

Na Seção 4.1 deste capítulo será apresentada uma visão geral da solução proposta, denominado *Analogus*. Em seguida, na Seção 4.2, serão descritos os principais artefatos para construção do ambiente *Analogus*. Já na Seção 4.3 serão descritas brevemente as tecnologias empregadas na implementação do ambiente. As Seções 4.4 e 4.5 apresentarão detalhes sobre a implementação do sistema de raciocínio baseado em casos e do agente inteligente de diálogo, respectivamente. Logo após, na Seção 4.6, serão apresentados os detalhes sobre a integração dos sistema de raciocínio baseado em casos e o agente inteligente de diálogo. Por fim, na Seção 4.7, serão apresentadas as telas da interface do ambiente *Analogus*.

## 4.1 Visão Geral da Solução Proposta

Para minimizar as dificuldades dos alunos iniciantes na resolução de problemas de programação, por meio de analogias, e tratar a carência de ferramentas que os auxiliem no emprego dessa forma de raciocínio, este trabalho propõe o *Analogus* [SJCF08; SJFC09].

O *Analogus* é um ambiente virtual de resolução de problemas de programação capaz de auxiliar o aluno iniciante a recordar problemas resolvidos, similares ao atual, ao ponto que possibilita a reflexão do aluno sobre os aspectos de semelhanças entre tais problemas.

Para implementação da solução foi realizada a integração de um sistema de raciocínio baseado em casos, para o domínio de programação, e um agente inteligente de diálogo – que simula um professor virtual em conversas sobre a utilização de padrões de programação para alunos iniciantes, sobre a sintaxe da linguagem de programação ensinada em sala de aula e sobre os aspectos de similaridades entre os problemas de programação. Na Figura 4.1, está ilustrada a visão geral da solução proposta.


Figura 4.1: Visão Geral da Solução Proposta

Em suma, no momento em que um novo problema de programação é passado como uma atividade para o aluno, o sistema de raciocínio baseado em casos recebe um conjunto de informações sobre tal problema e recupera automaticamente da memória de casos uma coleção de problemas previamente resolvidos pelo aluno, similares ao novo problema. Uma vez recuperados, os problemas similares são enviados ao agente inteligente de diálogo que conversa com o aluno sobre tais problemas e sobre os aspectos que fazem esses serem semelhantes.

Diante dos problemas recuperados e do diálogo com o *chatterbot*, o aluno cria a solução do novo problema – reutilizando o conhecimento das situações passadas – e a submete para a avaliação do professor que, por sua vez, efetua comentários sobre a solução. Assim que todas as considerações do professor são atendidas, a resolução do problema é finalizada e armazenada na memória de casos do *Analogus*.

Para atender aos aspectos supracitados e às características discutidas na Seção 3.6 do Capítulo 3, o *Analogus* foi desenvolvido seguindo a arquitetura cliente-servidor. Os detalhes arquiteturais e tecnológicos do *Analogus* serão destacados nas Seções 4.2 e 4.3, respectivamente.

Vale ressaltar que o *Analogus* é indicado para alunos iniciantes na resolução das atividades práticas das disciplinas de introdução à programação, principalmente, nos cursos que visam torná-los bons resolvedores de problemas de programação. Destaca-se que, para tanto, é fundamental seu emprego aliado a uma metodologia de ensino de padrões para alunos ini-

ciantes.

Por fim, é importante destacar que a versão atual do *Analogus* possibilita a resolução de problemas de programação seguindo a linguagem *Python*. Tal linguagem foi escolhida devido ao crescente interesse da comunidade em aplicá-la nos cursos introdutórios de programação [GPjBS06; PM06; Old05], além de ser a linguagem adotada no Curso de Ciência da Computação da Universidade Federal de Campina Grande (UFCG), no ensino desta disciplina.

## 4.2 Artefatos

Nesta seção serão descritos os artefatos criados para o desenvolvimento do ambiente *Analogus*, quais sejam: os requisitos funcionais e os não-funcionais, a descrição dos atores do sistema e o projeto arquitetural do ambiente.

Nos requisitos funcionais serão apresentadas as funções a serem implementadas no ambiente neste trabalho. Em se tratando dos requisitos não-funcionais, serão descritas as condições de comportamento e restrições. Na descrição dos atores serão definidos os usuários do ambiente e suas atribuições. Por fim, o projeto arquitetural descreverá os elementos que compõem a arquitetura do *Analogus*.

### 4.2.1 Requisitos Funcionais

Diante da análise dos trabalhos relacionados e das características discutidas na Seção 3.6 do Capítulo 3 foram levantados os requisitos funcionais a seguir.

RF1 **Prática do ensino com padrões de programação para alunos iniciantes**: O ambiente deve auxiliar na prática dos conceitos de padrões de programação ensinados em sala de aula.

- RF1.1 *Manutenção dos padrões*: o ambiente deve permitir adicionar, remover e manter os padrões de programação.
- RF1.2 *Visualização dos padrões*: o ambiente deve possibilitar a visualização das informações sobre os padrões ensinados em sala de aula.
- RF1.3 *Instanciação dos padrões*: o ambiente deve permitir a instanciação os padrões na área de edição de codificação da solução do problema de programação.

RF2 **Resolução de problemas**: o ambiente deve permitir a manutenção dos problemas, como também, a visualização e o reuso de suas soluções.

RF2.1 *Manutenção dos problemas*: o ambiente deve permitir adicionar, remover e manter os problemas de programação.

RF2.2 *Visualização do problema atual*: o ambiente deve possibilitar a visualização das informações sobre o problema que está sendo resolvido.

RF2.3 *Visualização dos problemas resolvidos*: o ambiente deve possibilitar a visualização das informações sobre o problema que já foram resolvidos.

RF2.4 *Resolução do problema*: o ambiente deve permitir que um determinado problema seja resolvido.

RF2.4 *Submeter a solução*: o ambiente deve permitir que a solução de determinado problema seja submetida a avaliação.

RF2.5 *Avaliação da solução*: o ambiente deve permitir que a solução para um determinado problema seja avaliada, possibilitando a emissão dos devidos comentários.

RF2.6 *Visualização dos comentários*: o ambiente deve possibilitar a visualização dos comentários emitidos sobre a solução para o problema.

RF2.7 *Execução do problema*: o ambiente deve permitir que o problema resolvido seja executado pelo interpretador Python.

RF3 **Recuperação dos problemas similares**: O ambiente deve recuperar da sua memória uma lista de problemas similares.

RF4 **Comunicação entre o RBC e o *chatterbot***: O ambiente deve possibilitar a troca de mensagens entre o sistema de raciocínio baseado em casos e o agente inteligente de diálogo.

RF5 **Diálogo com o agente**: O ambiente deve emitir trocas de mensagens entre o ator e o agente inteligente de diálogo.

### 4.2.2 Requisitos Não-Funcionais

Os requisitos não-funcionais levantados na análise da solução proposta são:

- **Dimensão da base de problemas**: A base de problemas deve oferecer ao aluno situações que englobem, além das listas de exercícios indicadas pelo professor, problemas extras para que o aluno possa praticar. Para tanto, é necessário ampliar a base de dados.

- **Tempo de resposta do ambiente**: O ambiente deve limitar seu tempo máximo de resposta em 30 (trinta) segundos para as atividades que necessitem da comunicação entre o cliente e o servidor.

- ***Interface* de fácil uso**: A *interface* do ambiente deverá possibilitar a consulta e utilização dos padrões de programação, dos problemas resolvidos e dos comentários com no máximo 3 (três) cliques.

### 4.2.3 Atores do Sistema

O *Analogus* possui 2 (dois) usuários distintos:

- **Professor:** este ator utiliza o sistema para adicionar, remover ou modificar os problemas e os padrões de programação, cadastrar os alunos da disciplina, corrigir as soluções dos problemas e emitir os comentários.

- **Alunos:** interagem com o sistema para resolução dos problemas de programação, diálogo com o professor virtual, visualização dos padrões e dos problemas resolvidos.

Na Figura 4.2, está ilustrado o diagrama de *use cases* (casos de uso) que descreve a interação entre os atores e os requisitos funcionais definidos.

### 4.2.4 Projeto Arquitetural

O *Analogus* se baseia na arquitetura cliente-servidor, na qual cada instância de um cliente envia requisições de dado para o servidor conectado e espera pela resposta. Por sua vez, o servidor recebe tal requisição, processa a informação e devolve o resultado para o cliente.

Para implementar essa arquitetura foi utilizado o *framework Google Web Toolkit (GWT)*, que facilita a implementação de aplicações *Web* seguindo essa estrutura arquitetural. Na Seção 4.3.1 está descrita com detalhes essa tecnologia, sendo apresentadas as suas principais vantagens.

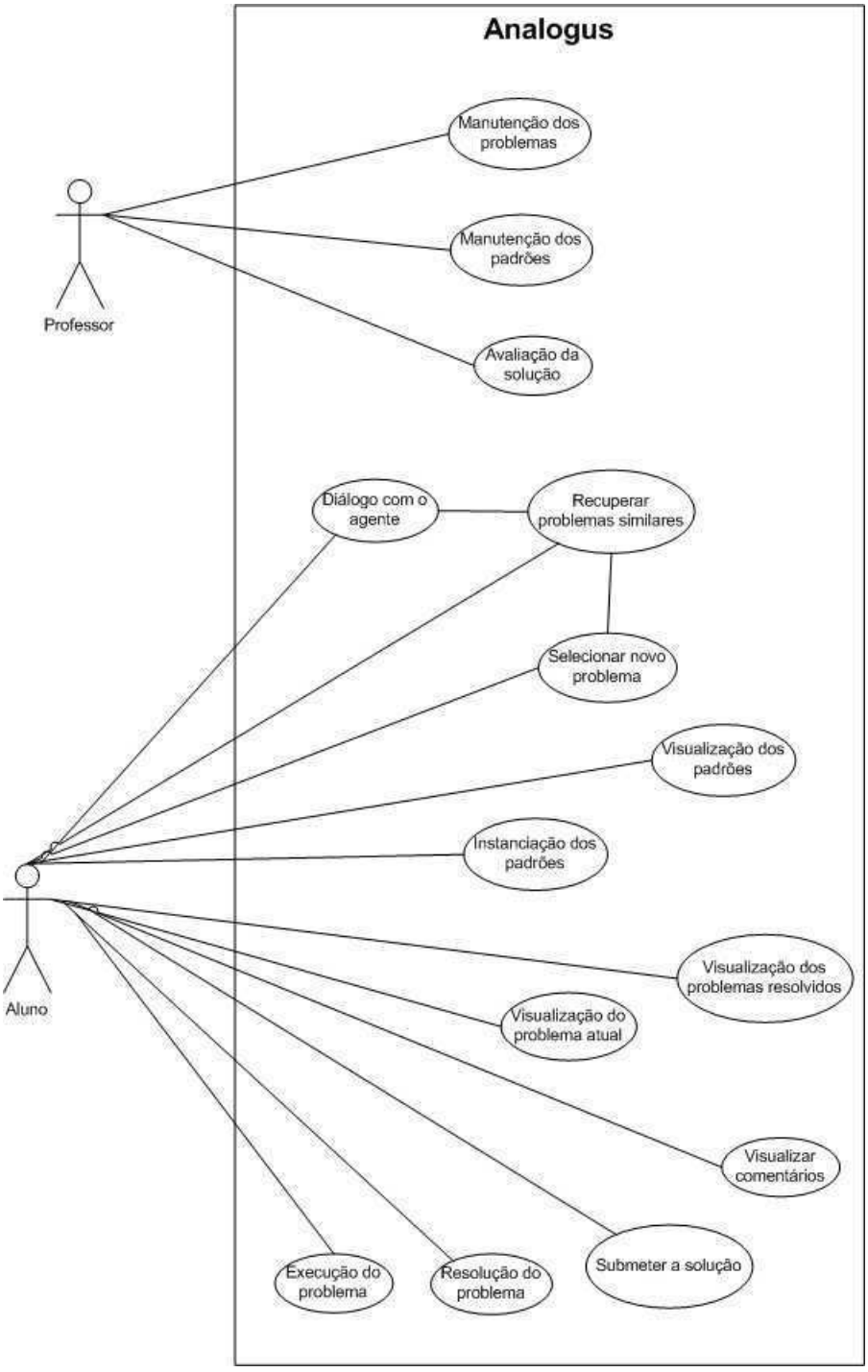



Figura 4.2: Diagrama de *use cases* do *Analogus*.

Dentro desse contexto, o lado cliente é dividido em 2 (duas) visões de acesso: a visão do aluno e a visão do professor. O lado servidor é constituído por 2 (dois) módulos – o módulo de resolução de problemas e o módulo de cadastro, ambos desenvolvidos neste trabalho –, 2 (dois) *frameworks* de terceiros e as bases de dados e de conhecimento. Os *frameworks* de

terceiros foram utilizados para agilizar o desenvolvimento e estão descritos em detalhes na Seção 4.3. Na Figura 4.3, está ilustrado o projeto arquitetural do *Analogus*.

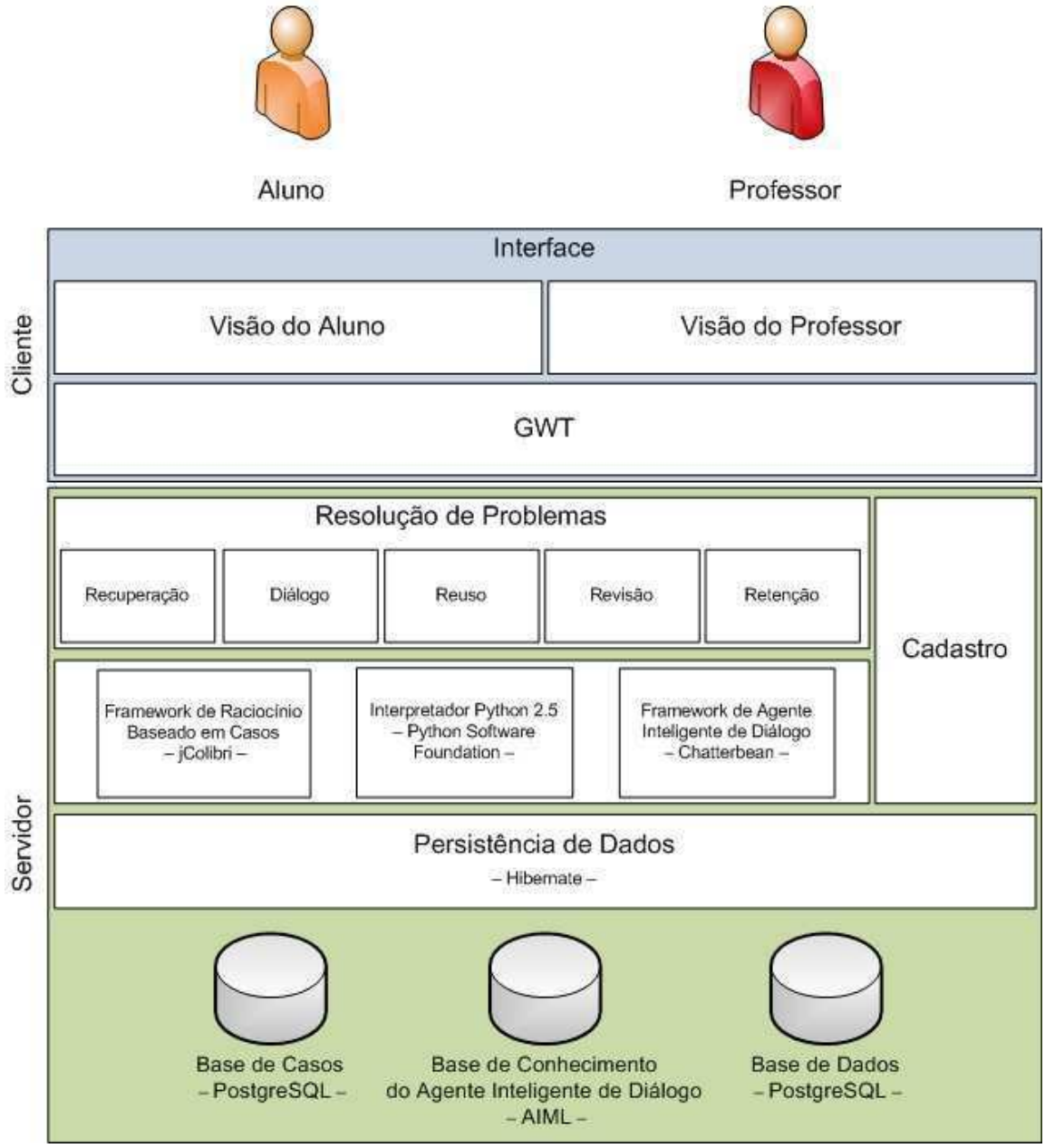



Figura 4.3: Projeto arquitetural do ambiente Analogus.

De acordo com o projeto arquitetural (Figura 4.3), no lado cliente do *Analogus*, a visão do aluno é representada por uma interface *Web* que lhe permite visualizar o problema a ser resolvido, consultar os problemas resolvidos e dos padrões de programação, dialogar com o professor virtual, solucionar o problema e executá-lo, submeter a solução para o professor e analisar os comentários por ele emitidos.

A visão do professor permite que este analise os problemas solucionados pelos alunos e emita seus comentários e, ainda, mantenha as informações do ambiente – problemas e padrões de programação.

Do lado servidor, o módulo de resolução de problemas realiza a integração do sistema de raciocínio baseado em casos e do agente inteligente de diálogo para resolução de problemas de programação no ambiente *Analogus*.

O módulo de cadastro gerencia a manutenção dos problemas, os assuntos da disciplina, a categoria do problema e os padrões de programação. Neste, o professor pode inserir, remover e editar cada uma das informações citadas.

O *framework* para criação de sistemas de raciocínio baseado em casos – *jColibri* (http://gaia.fdi.ucm.es/projects/jcolibri/) – foi utilizado para a instanciação de um RBC para auxiliar na resolução de problemas de programação. Para isso, foi necessário modelar o caso nesse domínio, definir os índices e os atributos, analisar as funções de similaridade locais, ajustar os pesos para cada atributo e, por fim, alimentar a base com um conjunto de casos iniciais (os exemplos e problemas resolvidos pelo professor em sala de aula). Nas Seções 4.3.2 e 4.4 estão descritos, com detalhes, o *framework* e sua instanciação no *Analogus*, respectivamente.

Para construção do professor virtual foi utilizado o *framework Chatterbean* (http://chatterbean.bitoflife.cjb.net/). No *Analogus* essa biblioteca tem a função de inferir uma resposta diante do questionamento do aluno ou de uma mensagem enviada pelo módulo de resolução de problemas. Esse *framework* acessa diretamente a base de conhecimento do agente inteligente de diálogo, implementada seguindo a linguagem AIML. Nas Seções 4.3.3 e 4.5 estão descritos, com detalhes, o *framework* e sua instanciação no *Analogus*, respectivamente.

Outra ferramenta de terceiros utilizada é o interpretador Python. Esse interpretador foi adicionado à arquitetura do *Analogus* para permitir que os alunos executem suas soluções. Para tanto, o módulo de resolução de problemas salva no servidor uma versão temporária da solução do aluno, requisita ao intepretador que execute essa solução e devolve o resultado para o aluno.

Para persistência de dados, foi utilizada a tecnologia *Hibernate* (https://www.hibernate.org/), a qual é responsável pelo mapeamento objeto-relacional

dos dados. Assim, é possível implementar o sistema orientado a objetos e armazenar os dados em um banco de dados relacional.

O ambiente *Analogus* acessa 3 (três) bases distintas: a base de casos, a base de dados e a base de conhecimento do agente inteligente de diálogo. As duas primeiras foram implementadas no sistema de banco de dados relacional PostgreSQL, enquanto a base de conhecimento do agente é formada por um conjunto de arquivos no formato AIML. A seguir, uma descerião mais detalhada dessas bases.

- **Base de casos**: armazena os casos – problemas resolvidos pelo aluno – do sistema de raciocínio baseado em casos;
- **Base de dados**: armazena as informações cadastradas pelo professor como, por exemplo, os problemas a serem solucionados, os padrões de programação, as categorias dos problemas e os assuntos apresentados em sala de aula;
- **Base de conhecimento do agente**: armazena o conhecimento do agente para que o *framework Chattebean* possa inferir uma resposta para as perguntas realizadas. Informações detalhadas sobre tal *framework* serão apresentadas na Seção 4.3.3.

## 4.3 Tecnologias Utilizadas

O *Analogus* foi implementado utilizando a linguagem *Java*, versão 6 (http://java.sun.com/javase/). A seguir, as justificativas pela escolha desta linguagem.

- suporte à orientação a objetos;
- gratuidade de ambientes de desenvolvimento integrado, como o *Eclipse IDE* (http://www.eclipse.org);
- existência de bibliotecas para o desenvolvimento *Web*, como o *Google Web Toolkit*;
- disponibilidade de *framework* para implementação do sistema de raciocínio baseado em casos – *jColibri* – e do agente inteligente de diálogo – *Chatterbean*.

Na implementação foi utilizado o *Eclipse IDE*, na versão 3.3, e adicionados os *plugins Cypal Studio* (http://www.cypal.in/studio) e o *Subclipse* (http://subclipse.tigris.org/).

Este último como *front-end* para o controle de versões usando o *Subversion* (http://subversion.tigris.org/). Foi utilizado também o *Cypal Studio*, para facilitar o desenvolvimento de aplicações usando o *framework GWT*.

### 4.3.1 *Google Web Toolkit - GWT*

O *Google Web Toolkit* (http://code.google.com/webtoolkit) é um *framework open-source*, sob termos da licença *Apache 2.0*, para o desenvolvimento eficiente de aplicações *web*, usando apenas a linguagem de programação *Java*.

Essa biblioteca permite que o desenvolvedor implemente toda a aplicação em *Java* e, em seguida, a compile de forma transparente e otimizada para os navegadores *IE*, *Firefox*, *Mozilla*, *Safari* e *Opera*. É possível ainda reutilizar componentes visuais na *interface* da aplicação e testá-la usando o *framework JUnit*.

As aplicações utilizando o *GWT* seguem a arquitetura cliente-servidor, na qual o cliente é capaz de invocar um método no servidor. Nesse sentido, essa biblioteca facilita o desenvolvimento serializando automaticamente os argumentos, invocando o método adequado no servidor e desserializando o valor de retorno para o código cliente.

### 4.3.2 *jCOLIBRI*

O *jCOLIBRI* (http://gaia.fdi.ucm.es/projects/jcolibri/) [JAAP05; Gar08] é um *framework open-source*, orientado a objetos, em *Java*, criado pelo *GAIA* (*Group For Artificial Intelligence Applications*) da Universidade *Complutense* de *Madrid*, para o desenvolvimento de sistemas de Raciocínio Baseado em Casos.

O *framework jCOLIBRI* reúne um conjunto de classes abstratas e interfaces gráficas para o desenvolvimento de aplicações de RBC. Essa biblioteca baseia-se no ciclo básico proposto por Aamodt e Plaza (1994) e permite a definição de casos, da estrutura da base de casos e das funções de similaridade global e local.

O *GAIA* disponibiliza uma vasta documentação sobre a biblioteca como, por exemplo, *javadoc*, tutorial, exemplos de código e artigos acadêmicos.

Recentemente, o *jCOLIBRI* foi considerado a principal ferramenta para o desenvolvimento de sistemas de raciocínio baseado em casos pela revista mi+d (http://www.madrimasd.org), após atingir 5.000 *downloads* distribuídos em 70 países, es-

palhados pelos continentes.

### 4.3.3 *ChatterBean*

O *ChatterBean* (http://chatterbean.bitoflife.cjb.net/) é um interpretador *AIML*, sob os termos da licença *GNU GPL*, que permite o desenvolvimento de agentes inteligentes de diálogo seguindo a arquitetura do ALICE, descrita na Seção 2 do Capítulo 2.4. Dentre as principais características do *Chatterbean*, destacam-se:

- compatibilidade com a linguagem *AIML* na versão 1.0;
- totalmente desenvolvido em *Java*;
- código documentado, acompanhado de uma série de testes de unidade e de aceitação.

## 4.4 Implementação do Sistema de Raciocínio Baseado em Casos para Resolução de Problemas de Programação

Para implementação do sistema de raciocínio baseado em casos para resolução de problemas de programação, utilizando o *framework jColibri*, foi necessário realizar algumas atividades, sendo essas:

- Modelar o caso para o domínio de programação, definindo os índices e os atributos do caso;
- Definir as funções de similaridade local e global;
- Implementar uma função de pré-processamento de dados.

A seguir, será detalhada cada uma dessas etapas realizadas na implementação desse sistema de raciocínio baseado em casos.

### 4.4.1 Representação do Caso

O caso, para um sistema de raciocínio baseado em casos, é um registro de uma experiência armazenada na memória, que é comumente representado pela descrição de um problema e da forma como este foi solucionado.

No contexto de resolução de problemas de programação, o caso pode ser representado pelo problema de programação e pela solução algorítmica empregada pelo programador para solucioná-lo.

Para este sistema, o problema de programação é descrito da seguinte forma: pelo enunciado, pelos padrões de programação sugeridos pelo professor para solucioná-lo, pela complexidade e pela categoria do problema. A solução, por sua vez, é o algoritmo desenvolvido pelo aluno para solucionar tal problema. O diagrama de classes apresentado na Figura 4.4, ilustra esta representação do caso.

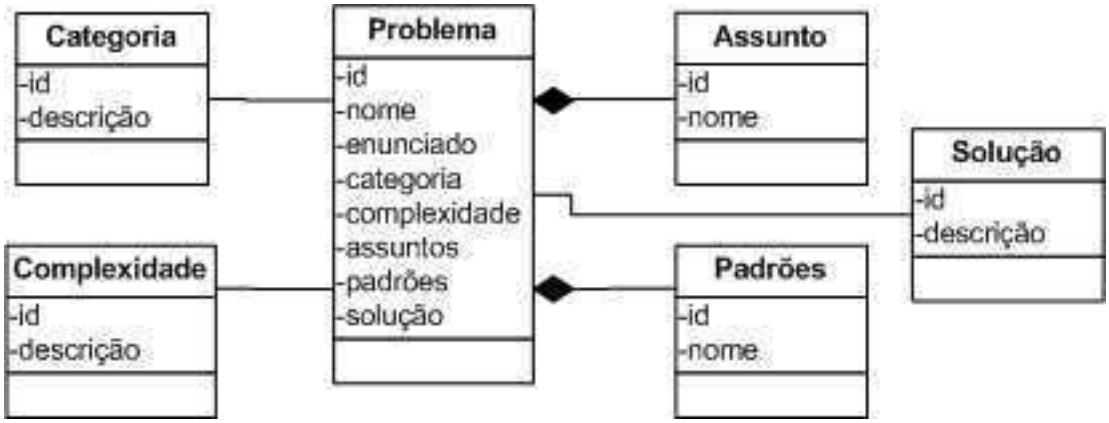



Figura 4.4: Representação do caso.

Para o enunciado do problema, é executado um pré-processamento, com intuito de extrair desse as palavras comuns ao domínio. Tal processamento é necessário para tornar a similaridade entre duas *strings* mais significativa. Essa função foi desenvolvida neste trabalho e é executada pelo sistema de raciocínio baseado em casos na fase de recuperação. No Apêndice A está descrita uma lista de palavras comuns ao domínio de programação utilizadas neste trabalho.

Os padrões de programação foram adicionados na representação do caso, uma vez que um problema pode ser solucionado a partir da combinação de padrões, aninhando-os ou intercalando-os. Além disso, diversos trabalhos, conforme discutido na Seção 2.3 do Capítulo 2, indicam a utilização desses padrões como uma alternativa para empregar o raciocínio por analogia no ensino de programação para alunos iniciantes.

Neste trabalho foram utilizados 15 (quinze) padrões de programação para iniciantes: 5 (cinco) padrões elementares e 10 (dez) padrões algorítimicos, como mostrado na Tabela 4.1. Tais padrões foram definidos no trabalho de Orna Muller [Mul05] sobre ensino de

programação utilizando padrões e no ProPAT [Del05; dBADM05].

Tabela 4.1: Lista de Padrões para Alunos Iniciantes em Programação.

| **Nome** | **Descrição** | **Padrão** |
|---|---|---|
| Algum valida | Verifica se ao menos um elemento da coleção atende a uma determinada condição. | Algorítmico |
| Buscar item | Busca por itens em uma coleção. | Algorítmico |
| Par | Indica se um número é par ou impar. | Algorítmico |
| Primo | Indica se um número é primo. | Algorítmico |
| Inverter a posição do item | Inverte a posição dos elementos em uma coleção. | Algorítmico |
| Lista ordenada | Verifica se uma coleção está ordenada. | Algorítmico |
| Ordenar lista | Ordena uma coleção. | Algorítmico |
| Repetição com indicador de passagem | Padrão de repetição em que ao fim do *loop* uma variável indica se uma determinada condição foi atendida. | Elementar |
| Seleção simples | Representa um comando de seleção simples. | Elementar |
| Seleção em conjunto | Representa uma seleção dentre de um conjunto de possibilidades. | Elementar |
| Repetição com sentinela | Repetição que utiliza um *flag* como condição de parada. | Elementar |
| Repetição contada | Repetição executada $n$ vezes. No qual, $n$ é a quantidade de repetições previamente conhecida. | Elementar |
| Todos validam | Verifica se todos os elementos da coleção atendem a uma determinada condição. | Algorítmico |
| Trocar a posição de um item | Troca a posição de um dado item em uma coleção. | Algorítmico |

Os níveis de complexidade dos problemas foram definidos como fácil, médio e difícil. A categoria classifica os problemas em matemáticos, de sistema de informação (SI) ou de jogos, seguindo a classificação de Medonça [Men08], a qual está descrita com detalhes na Tabela 4.2.

Tabela 4.2: Descrição do problema consultado no RBC.

| **Categoria** | **Descrição** | **Exemplos** |
|---|---|---|
| Matemáticos | Problemas relacionados ao domínio da matemática. | Fatorial, Equação de Segundo Grau, número primo, entre outros. |
| Jogos | Problemas de diferentes domínios e que se apresentam como jogos. | Torre de Hanoi, Missionários e Canibais, Forca, Damas, entre outros. |
| SI | Problemas que automatizam as atividades operacionais. | Controle de Pessoal, Controle de Linha de Produção, Controle de Estoque, etc. |

### 4.4.2 Funções de Similaridade

Na implementação desse sistema de raciocínio baseado em casos foi utilizado o algoritmo de recuperação do vizinho mais próximo, tendo a Equação 2.1 como função de similaridade global.

Para o cálculo da similaridade local dos atributos foram utilizadas as funções:

- **Equal:** devolve o valor 1 se os atributos possuem o mesmo valor. Caso contrário, devolve 0.
- ***Cosine Similarity*:** essa função computa a relação entre dois vetores a partir da Equação 2.3.

Na Tabela 4.3 é apresentada a função de similaridade local e o peso dado a cada atributo representado no caso.

Tabela 4.3: Similaridade local para os atributos do caso.

| **Atributo** | **Similaridade Local** | **Peso** |
|---|---|---|
| Categoria do problema | *Equal()* | 1.0 |
| Complexidade | *Equal()* | 1.0 |
| Enunciado do problema | *CosineCoefficient()* | 1.0 |
| Assuntos tratados | *CosineCoefficient()* | 5.0 |
| Padrões de programação | *CosineCoefficient()* | 1.0 |

A escolha das funções de similaridade local e o ajuste ideal dos pesos para cada atributo representado no caso, com intuito de aumentar a precisão da similaridade entre o novo problema e os problemas recuperados, foram obtidos por meio de experimentos. Detalhes sobre este estudo estão descritos na Seção 5.1 do Capítulo 4.

## 4.5 Implementação do Agente Inteligente de Diálogo para Resolução de Problemas de Programação

O agente inteligente de diálogo desenvolvido neste trabalho simula um professor virtual em conversas sobre a utilização de padrões de programação com alunos iniciantes, sobre a sintaxe da linguagem de programação ensinada em sala de aula e sobre os aspectos de similaridades entre os problemas de programação.

A implementação desse professor virtual segue a arquitetura do *ALICE* (*vide* Capítulo 2 Seção 2.4.2) e tem a base de conhecimento armazenada em um conjunto de arquivos no formato *AIML* (*vide* Capítulo 2 Seção 2.4.3).

O professor virtual do ambiente *Analogus* apresenta 4 (quatro) características marcantes, sendo essas:

- Discutir com o aluno sobre os aspectos de similaridade entre os problemas resolvidos;
- Esclarecer dúvidas sobre a linguagem de programação (Python) e os padrões ensinados em sala de aula pelo professor;
- Interface animada, representando o estado emocional do *chatterbot*. No *Analogus*, o *chatterbot* pode apresentar as expressões faciais de tristeza, de tranquilidade, de felicitação e de alegria;
- Interrupção do fluxo normal do diálogo quando o aluno apresenta dificuldades em identificar os problemas similares.

Desse modo, para implementação desse agente inteligente de diálogo utilizando o *framework Chatterbean* foi necessário realizar algumas atividades, sendo essas:

- Definir o fluxo do diálogo;
- Implementar estratégias de interrupção do fluxo normal do diálogo quando o aluno apresenta dificuldades em identificar os problemas similares;
- Criar a base de conhecimento do agente.
- Definir as estratégias emocionais do *chatterbot*.

Para discutir com o aluno sobre os problemas similares e suas características, a base de conhecimento do professor virtual é acessada pelo módulo de resolução de problemas e atualizada com informações sobre: Quais problemas são similares ao atual? Qual o nível de similaridade de cada atributo em relação ao problema atual (Categoria, Enunciado e Padrões de Programação)?

O professor virtual conversa sobre o grau de similaridade de forma discretizada. Desse modo, foram criados 5 (cinco) níveis de similaridade: idênticos, muito semelhantes, semelhantes, pouco semelhantes e diferentes. Na Tabela 4.4 estão descritos os limites para cada um dos níveis de similaridade.

Tabela 4.4: Similaridade local para os atributos do caso.

| **Níveis** | **Grau de similaridade (x)** |
|---|---|
| Idênticos | $80\% \leq x \leq 100\%$ |
| Muito semelhantes | $60\% \leq x < 80\%$ |
| Semelhantes | $40\% \leq x < 60\%$ |
| Pouco semelhantes | $20\% \leq x < 40\%$ |
| Diferentes | $0\% \leq x < 20\%$ |

Para exemplificar o diálogo entre o professor virtual e o aluno, no Código-Fonte 4.1 está descrito um trecho de diálogo entre o professor virtual e uma aluna. Nessa conversa, a aluna está tentando resolver o Problema A e o professor virtual está lhe ajudando na identificação dos problemas similares.

Código Fonte 4.1: Exemplo de diálogo do *Chatterbot*.

```
1  Professor: Olá minha cara aluna! Vamos resolver o Problema A? (SIM/NÃO)
2       Aluno: Sim
3  Professor: Ótimo! Nesse problema temos que 'Escreva um programa que
       imprime uma tabela em que cada linha contém: um inteiro e sua potência
        de 2. O número de linhas a produzir deve ser pedido ao usuário. Os
       números da primeira coluna devem estar justificados à esquerda e as
       potências à direita. Entre as duas colunas deve haver apenas um espaço
        em branco.'. Você se recorda de ter resolvido algum problema
       semelhante ao Problema A? (SIM/NÃO)
4       Aluno: Sim
5  Professor: Então, por favor, selecione na janela de 'Problemas Resolvidos
       ', um problema que você julga semelhante ao Problema A.
6  Professor: Hum... Você selecionou o problema: 3. Vamos analisar as
       semelhanças entre o problema atual e o selecionado. Quais
       características você julga semelhante entre o Problema A e o 3? (
       ENUNCIADO, CATEGORIA DO PROBLEMA, PADRÕES DE PROGRAMAÇÃO).
7       Aluno: Enunciado do Problema
8  Professor: Vejamos os enunciados dos problemas... Interessante!!! Pelo
       enunciado eu considero que estes problemas são 'bem semelhantes'.
9  Professor: Alguma outra característica você julga semelhante entre o
       Problema A e o Problema 3? (CATEGORIA DO PROBLEMA, PADRÕES DE
       PROGRAMAÇÃO).
10      Aluno: Os padrões de programação.
11 Professor: Quais padrões você acredita serem utilizados na solução de
       ambos os problemas?
12      Aluno: Padrão de repetição contada.
13 Professor: Parabéns!!! Ambos os problemas utilizam o padrão de repetição
       contada na sua solução.
```

As expressões faciais do *chatterbot* variam no decorrer do diálogo. Quando o *bot* requisita o aluno para identificar um problema similar e o aluno, por sua vez, aponta um problema pouco similar, o professor virtual muda sua expressão facial para triste, devolvendo a feição de tranquilidade assim que o aluno seleciona um problema similar. Por outro lado, quando o aluno identifica um problema similar ou seleciona características que apontam que os problemas são semelhantes, ao professor virtual atribui a expressão facial de felicidade. Na Figura 4.5, estão ilustradas as expressões faciais do professor virtual.

Figura 4.5: Expressões faciais do Professor Virtual.

Quando o aluno comete sucessivas seleções de problemas resolvidos, demonstrando a dificuldade em identificar problemas realmente similares, o professor apresenta a expressão facial de tristeza e ativa o modo de estratégias de interrupção do fluxo normal do diálogo.

O modo de interrupção do fluxo normal diálogo é ativado quando o aluno seleciona mais de 3 (três) problemas distintos, com grau de similaridade em relação ao problema atual "pouco semelhantes"ou "diferentes". Nesse momento, o módulo de resolução de problemas envia uma mensagem para o professor virtual informando que o aluno está sentindo dificuldades em identificar os problemas similares. O professor virtual, por sua vez, questiona ao aluno se ele está realmente sentindo dificuldades e se necessita de ajuda. Caso o aluno confirme a dificuldade, o professor virtual lhe aponta o problema mais similar ao atual.

A base de conhecimento é formada por 3 (três) arquivos no formato *AIML* e 1 (um) no formato *XML*, sendo esses:

- **python.xml:** conhecimento sobre a *sintaxe* da linguagem *Python*;
- **padroes.xml:** conhecimento sobre os padrões de programação para alunos iniciantes;
- **alice.xml:** conhecimento geral do professor virtual. Criado para tornar o diálogo mais fácil, a partir de expressões de agradecimento, de cumprimento e de respostas padrões, emitidas quando não existe resposta para pergunta efetuada pelo aluno.

- ***context.xml*:** esse arquivo foi criado seguindo o formato *XML*. Tal arquivo é responsável por armazenar as últimas respostas do professor virtual e armazenar o conhecimento sobre a similaridade entre problemas de programação.

No Apêndice B está descrita uma parte da base de conhecimento do professor virtual.

## 4.6 Integração RBC com o Agente Inteligente de Diálogo para Resolução de Problemas de Programação

O módulo de resolução de problemas tem a função de controlar a aplicação e integrar o sistema de raciocínio baseado em casos e o agente inteligente de diálogo para resolução de problemas de programação ao ambiente *Analogus*. Para tanto, o módulo foi subdividido em 5 (cinco) módulos menores, sendo esses:

- **Recuperação**:
  - extrai as informações do problema selecionado pelo aluno;
  - consulta o sistema de raciocínio baseado em casos para recuperar os problemas resolvidos similares ao selecionado pelo aluno;
  - armazena as informações dos problemas recuperados na base de conhecimento do agente inteligente para possibilitar o diálogo do professor virtual com o aluno sobre a analogia entre os problemas.
- **Diálogo**:
  - gerencia a troca de mensagens entre o aluno e o professor virtual;
  - utiliza estratégias pedagógicas no diálogo com o aluno como, por exemplo, quando o professor virtual pede ao aluno para este selecionar um problema resolvido semelhante ao que está resolvendo e o aluno, por sua vez, seleciona mais de 3 (três) problemas distintos ou seleciona um problema pouco semelhante. Esse módulo interrompe o diálogo e envia uma mensagem ao professor virtual dizendo que o aluno está sentindo dificuldades na identificação dos problemas similares.
- **Reuso**:

  - gerencia a troca de mensagens entre o interpretador da linguagem *Python* localizado no servidor;
  - armazena temporariamente a solução do aluno no servidor para ser executado pelo interpretador;
  - permite que o aluno submeta a solução para revisão.

- **Revisão**:
  - permite que o professor revise as soluções criadas e realize os comentários.

- **Retenção**:
  - permite que a solução final do aluno seja armazenada.

Para facilitar o entendimento do funcionamento do *Analogus*, destacando as características do módulo de resolução de problemas, foi criado um conjunto de diagramas de interação, descrevendo a sequencia de atividades das principais funcionalidades do ambiente. A seguir, serão descritos os cenários e os diagramas criados.

Quando o aluno iniciante deseja selecionar um novo problema a ser resolvido e seleciona no menu "Abrir problemas"a visão do aluno, a fase de recuperação do RBC é iniciada. Para tanto, o módulo de resolução de problemas carrega os problemas cadastrados pelo professor e exibe suas informações em uma janela de diálogo para que o aluno selecione o problema que deseja solucionar.

Assim que o aluno seleciona o problema, o módulo de resolução de problemas, por meio do pacote de recuperação, cria uma consulta ao RBC com os dados do problema selecionado e requisita do *jColibri* a lista de problemas similares. Uma vez que os problemas similares ao consultado foram recuperados, o módulo de resolução atualiza o conhecimento do agente inteligente, salvando as informações dos problemas similares na base de conhecimento do agente. Logo em seguida, o módulo de resolução de problemas inicializa o professor virtual, iniciando a fase de Reuso do RBC. No diagrama de sequencia, ilustrado na Figura 4.6, está detalhado esse processo de seleção do problema para resolução.

Iniciada a fase de reuso, o professor virtual começa o diálogo com o aluno, seguindo o fluxograma ilustrado na Figura 4.7. Primeiramente, o professor virtual pergunta ao aluno se

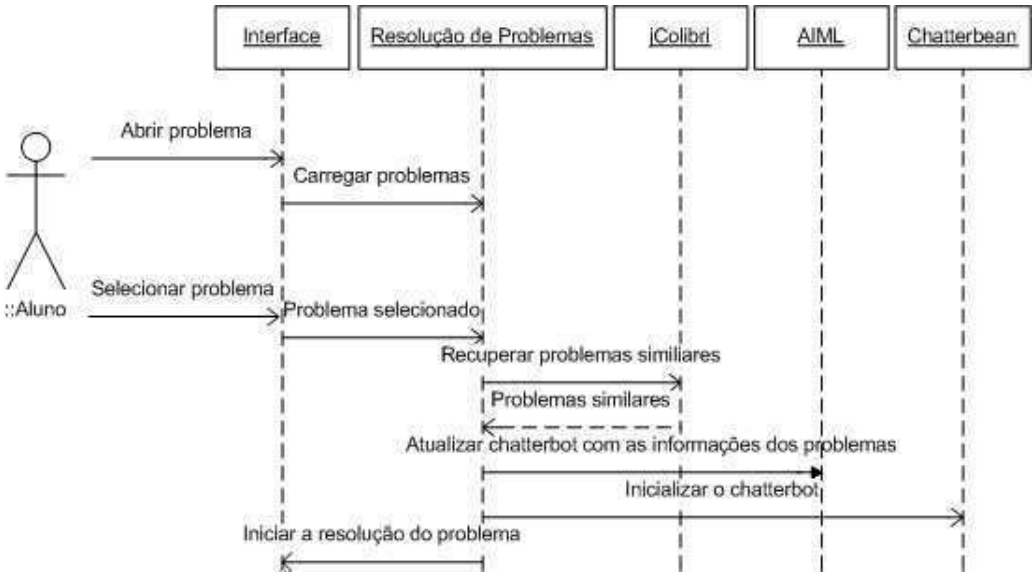



Figura 4.6: Diagrama de sequencia da seleção do problema para resolução.

esse se lembra de algum problema que resolveu semelhante ao novo problema. Caso o aluno se recorde, o professor pede que ele o selecione na interface e pergunta qual característica é semelhante entre os problemas – o lembrado e o atual.

Diante do problema selecionado e da característica informada pelo aluno, o professor virtual acessa sua base de conhecimento e informa o nível de similaridade entre tais problemas, para essa característica. O aluno, nesse momento, fica livre para analisar as características de semelhança entre esses 2 (dois) problemas, podendo escolher outro problema resolvido para analisar suas características, se julgar necessário.

Tal liberdade permite que, em algumas situações, o aluno selecione vários problemas distintos, aleatoriamente, com o intuito de investigar as características e semelhanças entre os problemas. Esse fato ocorre, principalmente, quando o aluno está sentindo dificuldades na identificação das similaridades. Nesta situação, o professor virtual interfere no fluxo do processo e questiona ao aluno se esse está sentindo dificuldades. Caso o aluno informe que está confuso, o professor recomenda ao aluno o problema mais similar no seu julgamento.

Assim, quando o aluno está seguro e apto a solucionar o problema, ele inicia a codificação, devolvendo o diálogo com o professor virtual sempre que necessário.

Desse modo, ao codificar a solução inicial para o problema, o aluno executa seu código para verificar se a solução está correta. Para tanto, seleciona em sua interface o menu "Executar solução". Em seguida, o módulo de resolução de problemas cria uma cópia temporária

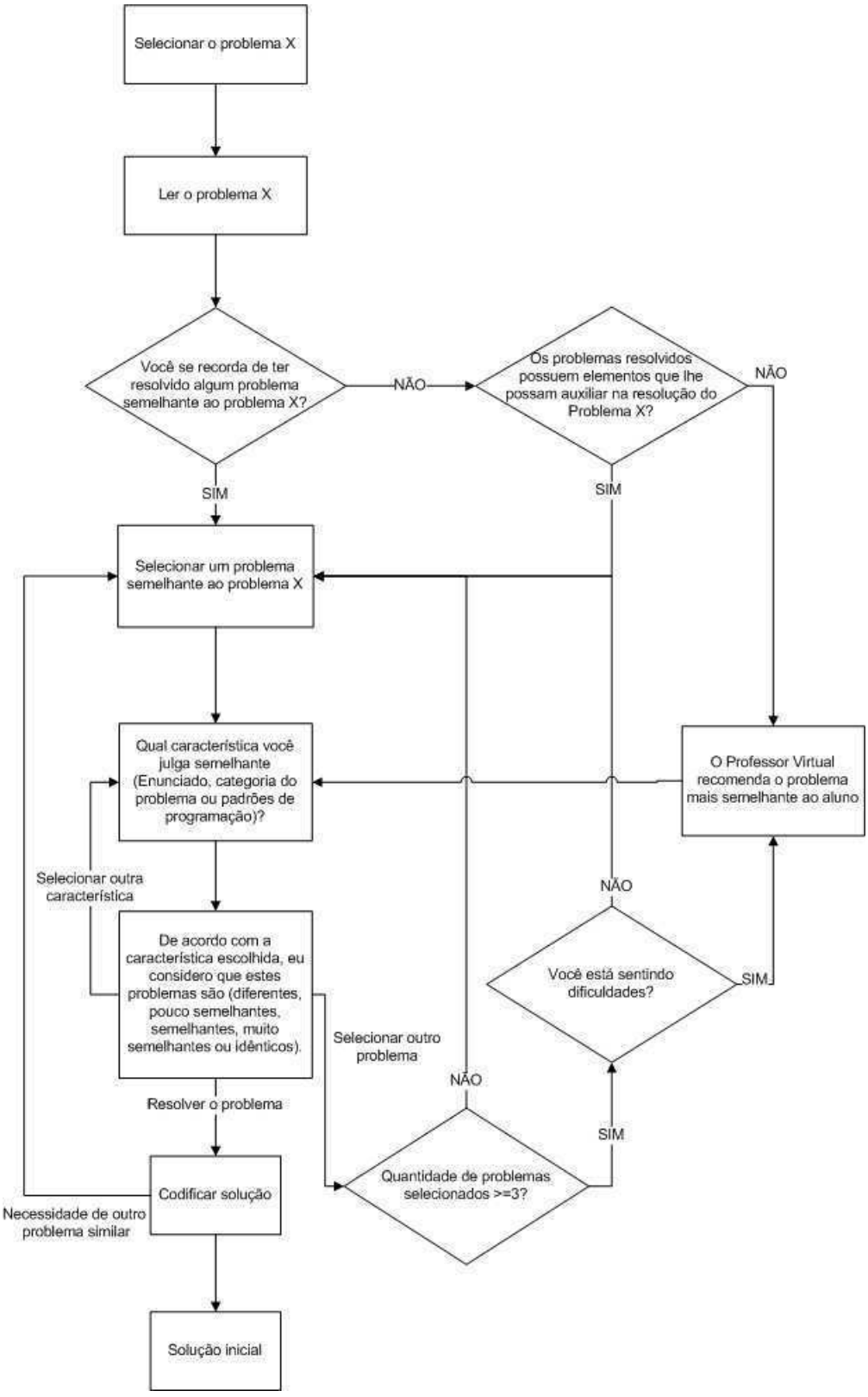



Figura 4.7: Fluxograma do diálogo entre o professor virtual e o aluno.

da solução do aluno no servidor e requisita ao interpretador de *Python* que execute a solução. A saída da execução é capturada pelo módulo de resolução de problemas e exibida para o aluno na interface.

Uma vez solucionado e executado o problema, o aluno submete a solução ao professor pelo menu "Submeter solução". O módulo de resolução de problemas marca o problema com o estado de avaliação, para que, posteriormente, o professor o avalie. No diagrama de sequencia, ilustrado na Figura 4.8, está descrita essa etapa do processo.

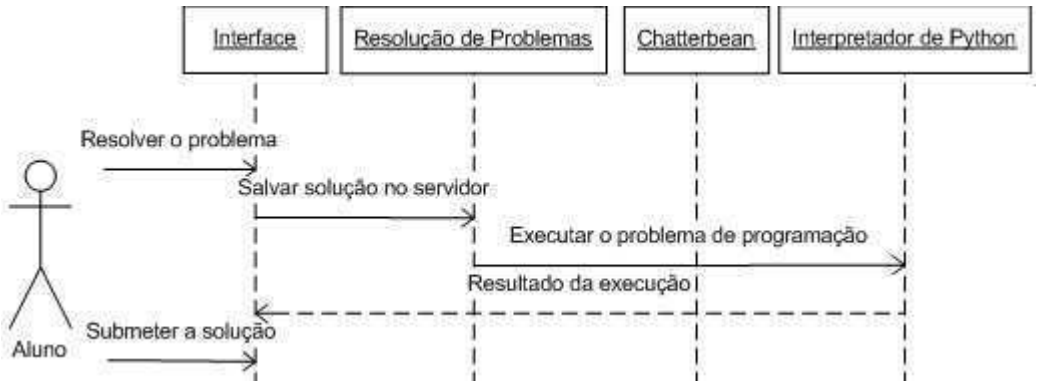



Figura 4.8: Diagrama de sequencia da execução da solução do aluno.

A visão do professor, por meio do menu "Exibir problemas resolvidos", permite que o professor visualize os problemas que já foram resolvidos pelos alunos e avalie as soluções submetidas. Para isso, o módulo de resolução de problemas recupera os problemas submetidos e exibe ao professor que, por sua vez, pode analisar o código, executar o programa e emitir os comentários sobre a solução. Tais comentários são armazenados e, posteriormente, visualizados pelo aluno. Quando a solução submetida é aprovada pelo professor, este a considera finalizada, executando a fase de retenção do ciclo de RBC. No diagrama de sequencia, ilustrado na Figura 4.9, estão descritas essas fases de revisão e retenção.

Uma vez finalizada a fase de retenção, o problema está solucionado e o aluno inicia a resolução de um novo problema de programação.

Mais detalhes sobre a implementação do sistema de raciocínio baseado em casos e do agente inteligente de diálogo serão apresentados nas Seções 4.4 e 4.5.


Figura 4.9: Diagrama de sequencia da retenção da resolução.

## 4.7 *Interface* do *Analogus*

O *Analogus* é constituído pela visão do aluno e do professor e sua interface apresenta um conjunto de janelas pré-fixadas, permitindo o acesso fácil, rápido e direto às principais funcionalidades do *Analogus*, principalmente na visão do aluno, que presa por uma *interface* bem simples para evitar a sua desconcentração.

As telas do *Analogus* permitem que o usuário redimensione, desloque e omita algumas das janelas para customizar a área de trabalho.

Nas seções 4.7.1 e 4.7.2, a seguir, são apresentadas as principais telas da visão do aluno e do professor, respectivamente.

### 4.7.1 Visão do Aluno

A interface da visão do aluno é constituída pelas seguintes janelas: de descrição do problema, de diálogo, de problemas resolvidos, de padrões de programação, de codificação, de console, de comentários e pela área de codificação.

Tais janelas permitem que o aluno escolha qual problema será resolvido; visualize o enunciado do problema, as soluções dos problemas resolvidos, os comentários do professor sobre a solução do problema atual e os padrões de programação ensinados em sala de aula; codifique a solução; execute seu código; e discuta sobre os aspectos de similaridade com o *chatterbot*. Na Figura 4.10, está ilustrada a *interface* do *Analogus*, na visão do aluno, no instante em que ele está selecionando um novo problema para resolver.

Uma vez selecionado, o enunciado do problema é exibido na janela de descrição, per-

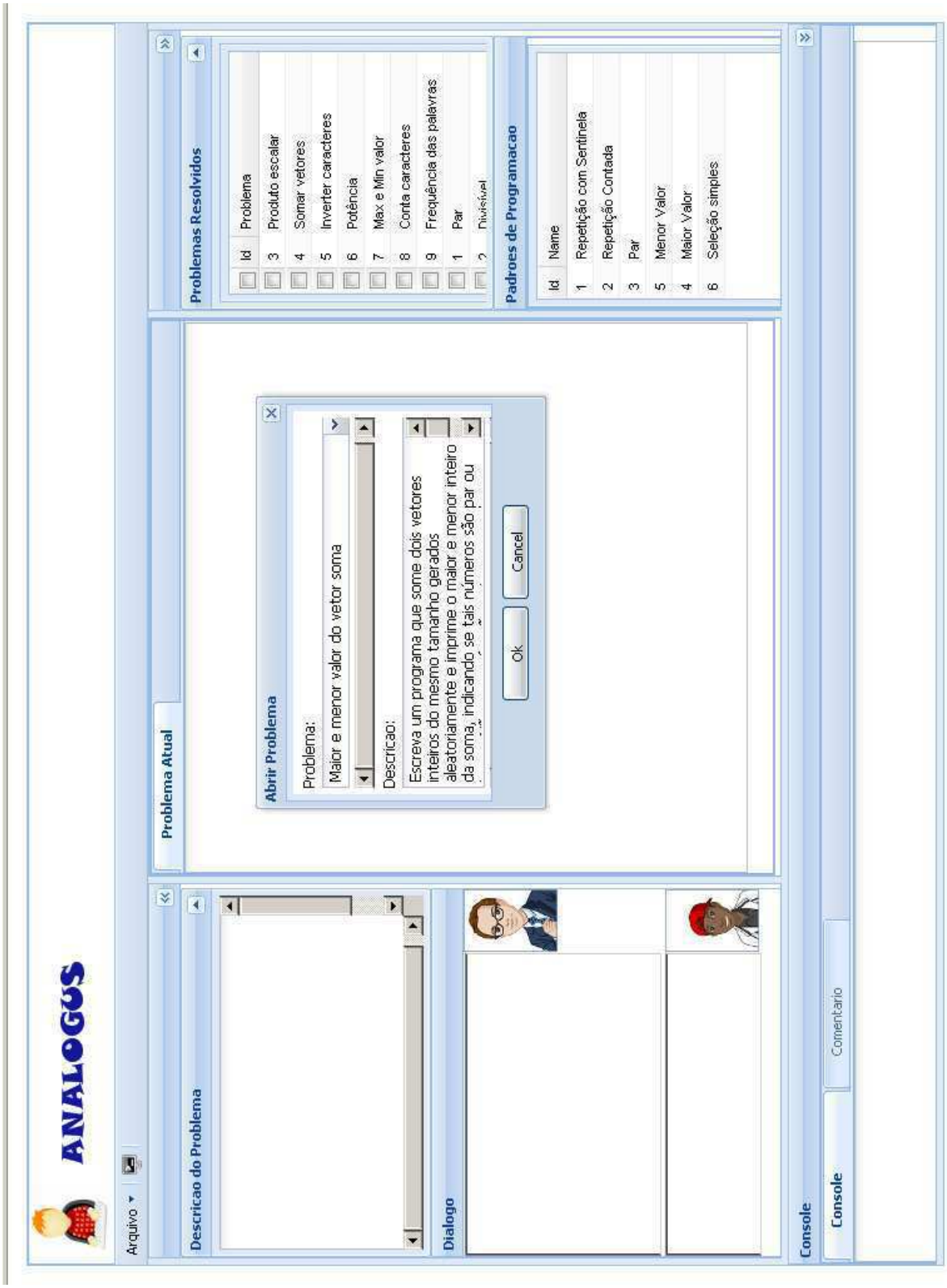



Figura 4.10: Abrir novo problema.

mitindo que o aluno o visualize a qualquer instante. A janela de diálogo permite que o aluno converse com professor virtual sobre os problemas similares, os padrões de programação e a sintaxe da linguagem, em qualquer momento.

O aluno, quando julgar necessário ou quando requisitado pelo professor virtual, pode selecionar e visualizar as informações dos problemas resolvidos em sua janela, conforme ilustrado na Figura 4.11.

Quando o aluno seleciona um problema resolvido, é criada na área de codificação uma nova aba com código da solução deste problema, como ilustrado na Figura 4.12. Dessa forma, o aluno pode visualizar, analisar e reutilizar as soluções dos problemas passados.

O aluno pode visualizar e instanciar um dos padrões listados na janela de padrões de programação. Quando o aluno seleciona um padrão, o *template* do problema, é inserido na área de codificação do problema atual. Na Figura 4.13, está ilustrado esse processo.

A *interface* do *Analogus* permite ainda que o aluno salve a resolução do problema, a submeta para avaliação ou execute o código a partir das opções da barra de *menu*. Quando a solução é executada, a saída do interpretador da linguagem *Python* é apresentada na janela de console. O aluno pode visualizar ainda os comentários emitidos pelo professor na janela de comentários.

### 4.7.2 Visão do Professor

A visão do professor é formada por um conjunto de telas de cadastro e pela *interface* principal que permite ao professor selecionar o aluno e todos os problemas por ele resolvidos, visualizar e executar a solução de um dado problema, consultar os padrões de programação ensinados e emitir comentários para o aluno sobre as soluções submetidas à avaliação. Na Figura 4.14, está ilustrada a visão do professor.

Esta interface é semelhante a do aluno. Entretanto, nesta há uma janela que lista todos os alunos cadastrados. Ao selecionar um aluno, o professor visualizará as soluções desse aluno na janela de problemas resolvidos.

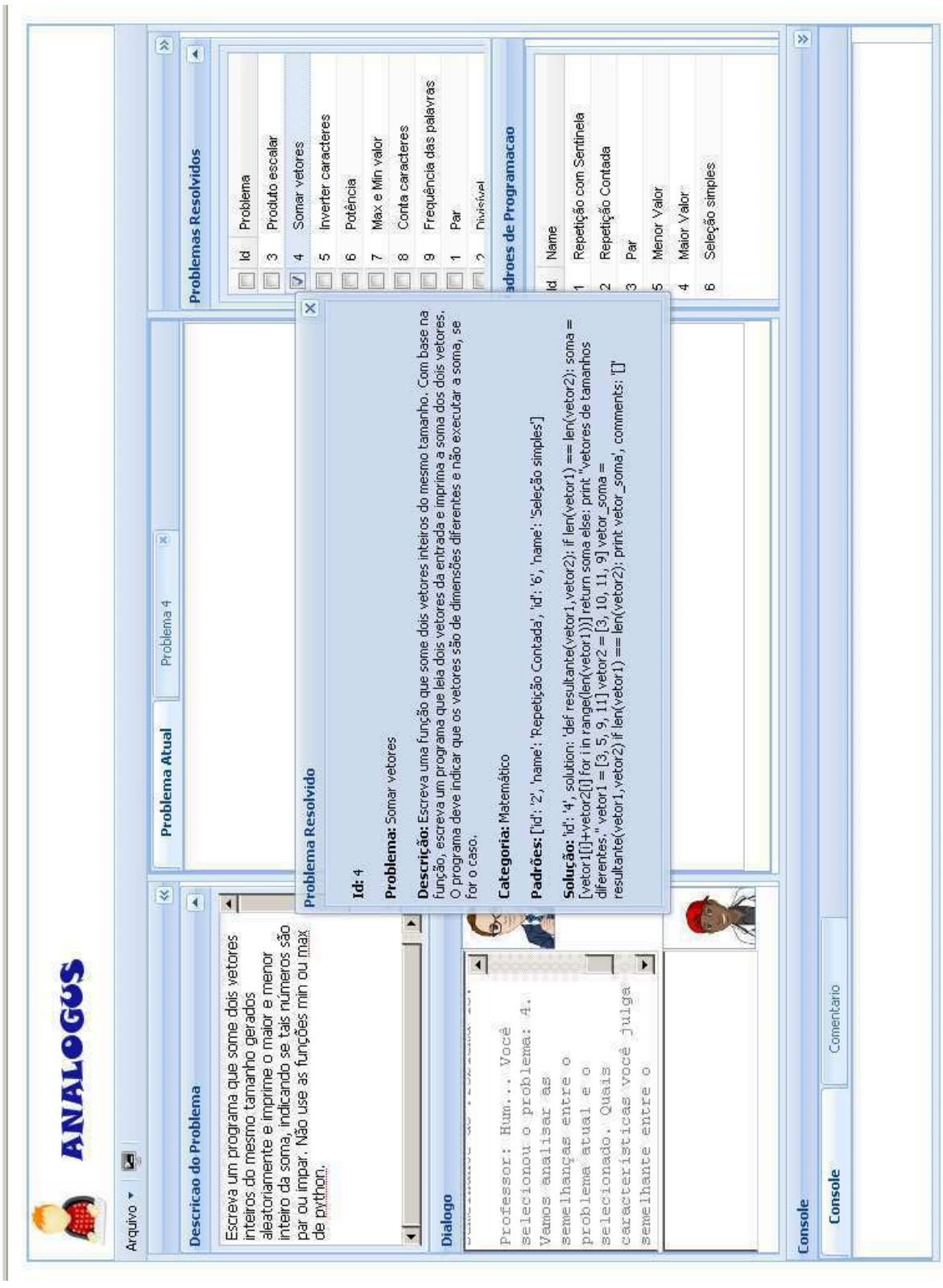



Figura 4.11: Selecionar os problemas resolvidos similares ao atual.


Figura 4.12: Visualizar o código dos problemas resolvidos.

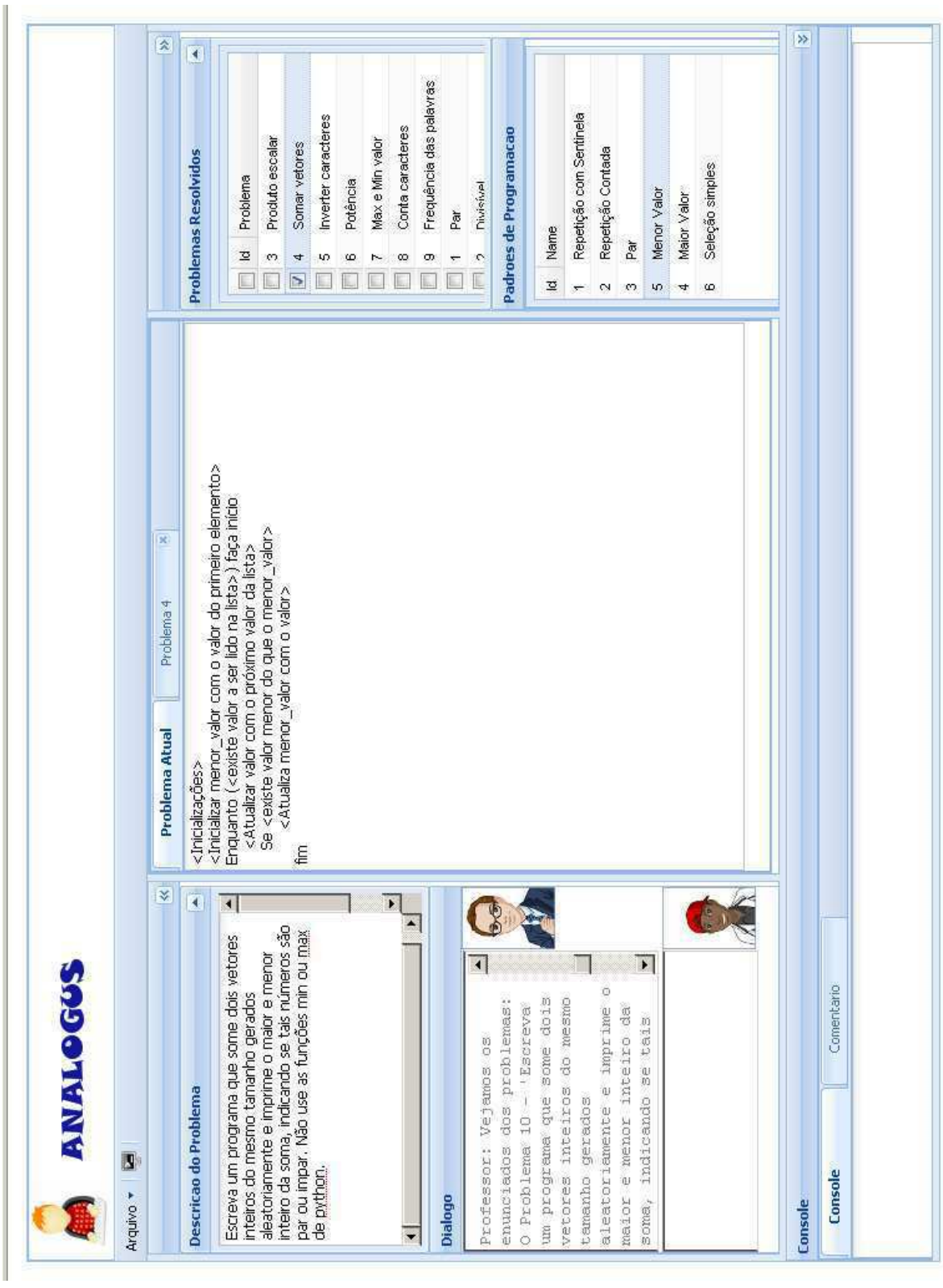



Figura 4.13: Inserir código *template* do padrão.


Figura 4.14: Tela da visão do professor.

# Capítulo 5

# Apresentação e Análise dos Resultados

Neste capítulo serão descritas as avaliações realizadas no ambiente *Analogus*. Inicialmente, na Seção 5.1, será apresentada a avaliação dos problemas similares recuperados pelo sistema de raciocínio baseado em casos. Em seguida, na Seção 5.2 serão apresentados e analisados os resultados de um ensaio experimental da utilização do *Analogus* por um conjunto de alunos do Curso de Ciência da Computação da Universidade Federal de Campina Grande.

## 5.1 Avaliação da recuperação de problemas de programação

Com o objetivo de avaliar o grau de similaridade dos problemas recuperados pelo RBC, para o domínio de programação, a fim de verificar a representação do caso e ajustar os pesos dos atributos, foi realizado um experimento com 200 problemas de programação extraídos de listas de exercícios de disciplinas de introdução à programação, obtidas na internet.

Os problemas foram divididos aleatoriamente em 2 (dois) conjuntos:

- **Conjunto de consulta:** representa os casos que serão utilizados como entradas para consulta ao RBC. No ambiente real, supondo que o aluno está utilizando o *Analogus*, esses problemas simbolizam os novos problemas a serem solucionados pelo aluno. Este conjunto é formado por 25% do total de problemas.

- **Memória de casos:** representa os casos que serão consultados pelo RBC, simbolizando os problemas previamente solucionados pelo aluno. A memória de casos é constituída pelos demais 150 dos problemas (75% do total).

Esse experimento simula a situação em que o aluno já possui alguma experiência, adquirida a partir dos exercícios resolvidos na disciplina.

Neste experimento, o valor do peso do enunciado do problema, dos assuntos tratados e dos padrões de programação variam de 1 a 5, sendo que o valor 5 indica o maior grau de importância do atributo. Para os demais atributos do caso – categoria e complexidade – convencionou-se o peso 1. Tal convenção ocorreu por motivos pedagógicos, visto que para o aluno é difícil identificar se dois problemas são similares levando em consideração apenas a categoria e a complexidade do problema, uma vez que estes atributos são bastante abstratos.

Na execução do experimento foram realizadas 50 (cinquenta) consultas ao sistema de raciocínio baseado em casos e, para cada consulta, os 5 (cinco) problemas mais similares ao consultado e suas respectivas configurações de pesos foram armazenados. Diante dos dados, foram calculados a média, o desvio padrão e o coeficiente de variação (C.V.)[1] das diferentes distribuições de peso.

Além da avaliação estatística realizada, os problemas recuperados pelo sistema eram analisados de forma subjetiva, a fim de verificar se existia alguma relação de semelhança das soluções entre o problema consultado e os indicados pelo RBC. Para isso, ao fim de cada experimento, eram analisados os problemas recomendados pelo sistema de raciocínio baseado em casos para os 5 (cinco) melhores resultados da análise estatística.

Foi utilizado o algoritmo do vizinho mais próximo para recuperação dos problemas similares. As funções de similaridade local utilizadas no experimento estão indicadas na Tabela 5.1 e descritas com detalhes na Seção 2.2.2 do Capítulo 2.

Tabela 5.1: Funções de similaridade local avaliadas no experimento.

| **Função** | **Enunciado** | **Assuntos da disciplina** | **Padrões** | **Complexidade** | **Categoria** |
|---|---|---|---|---|---|
| *Cosine Similarity* | X | X | X | | |
| *Dice's Coefficient* | X | X | X | | |
| *Overlap Coefficient* | X | X | X | | |
| *Jaccard Coefficient* | X | X | X | | |
| *Max String* | | | | X | X |
| *Equal* | | | | X | X |

[1]Medida de dispersão que compara diferentes distribuições. Essa função é calculada por meio da divisão do desvio padrão pela média e expressa em percentual.

### 5.1.1 Análise dos resultados

O experimento foi realizado em etapas, buscando sanar os pontos fracos identificados em cada uma delas. Assim, nesta seção serão descritos os resultados obtidos em cada uma das etapas e as medidas tomadas para otimizar a recuperação do RBC no domínio de programação.

#### Etapa 1

Nesta etapa, apenas as funções de similaridade local *Equal()* e *MaxString()* foram testadas. Para os atributos categoria e complexidade do problema, convencionou-se o uso da função *Equal()* e para os atributos do enunciado do problema, assuntos tratados e padrões de programação utilizou-se a função *MaxString()*. Na Tabela 5.2, estão descritos os 5 (cinco) melhores resultados dessa etapa do experimento.

Tabela 5.2: Melhores resultados obtidos com o ajuste de pesos na Etapa 1.

| Enunciado | Assuntos | Padrões | Categoria | Complex. | Média | Desvio Padrão | C.V. (%) |
|---|---|---|---|---|---|---|---|
| 1 | 5 | 1 | 1 | 1 | 0,66 | 0,18 | 26,68 |
| 1 | 5 | 2 | 1 | 1 | 0,62 | 0,18 | 28,48 |
| 2 | 4 | 1 | 1 | 1 | 0,60 | 0,17 | 28,38 |
| 2 | 5 | 1 | 1 | 1 | 0,63 | 0,18 | 27,87 |
| 2 | 5 | 2 | 1 | 1 | 0,59 | 0,17 | 29,30 |

Conforme observado na Tabela 5.2, os resultados com maiores médias e menores desvio padrão e coeficiente de variação foram obtidos para as configurações em que o atributo assunto tratado possuía valores de peso maiores do que os demais atributos. Isto comprova a importância deste atributo na identificação da similaridade entre os problemas. Tal fato pode ser justificado pedagogicamente, uma vez que problemas que tratam do mesmo assunto geralmente possuem um grande número de características em comum em sua resolução, o que afeta diretamente a similaridade entre os problemas.

Outro aspecto relevante refere-se ao fato de que os pesos obtidos para os atributos enunciado do problema e para os padrões de programação, de acordo com o coeficiente de correlação de Pearson[2], apontam indícios da existência de uma correlação entre o enunciado e o assunto, como também, entre os padrões e o assunto, uma vez que o valor $r = 0{,}40$. Para

[2]Mede o grau da correlação entre duas variáveis de escala métrica. O resultado da função é um valor entre -1 e 1, sendo que valores próximos de 0 indicam baixa correlação.

comprovação dessa correlação é necessário a realização de experimentos mais criteriosos, com uma base de problemas ainda maior.

Analisando estatisticamente os resultados, é possível observar ainda que as médias da similaridade entre os problemas são inferiores a 70%. Entretanto, os valores do desvio padrão (em torno de 0,18) e de coeficiente de variação (em torno de 30%) são elevados, indicando dispersão considerável entre os dados. Tais valores apontam, portanto, que, embora a memória de casos seja constituída de problemas com diferentes níveis de similaridade, as funções de similaridade local não estão conseguindo identificar essa semelhança ou a forma de representação escolhida não está adequada ao domínio.

Visualizando separadamente o resultado da similaridade dos atributos, foi possível concluir que era necessário um pré-processamento no enunciado do problema, isso porque, geralmente, o enunciado é um pequeno texto com palavras e expressões bastante comuns ao domínio como, por exemplo, "Faça um problema", "Crie uma função", "Leia"ou "Imprima". Desse modo, foi implementada uma função de pré-processamento para remover dos enunciados as *stopwords* e as expressões comuns ao domínio.

Assim, para exemplificar os resultados obtidos na recuperação dos problemas de programação, por meio do raciocínio baseado em casos nesta etapa, na Tabela 5.3 é apresentada a descrição de um problema que foi consultado.

Tabela 5.3: Descrição do problema consultado no RBC.

| **Enunciado** | **Comp.** | **Categoria** | **Assuntos** | **Padrões** |
|---|---|---|---|---|
| Escreva um programa em PASCAL para ler um número inteiro N e imprimir os N primeiros números primos. | Fácil | Matemático | Operadores, variáveis, comandos de entrada e saída, comandos de seleção e comandos de repetição. | Primo e Repetição contada |

A partir dessa consulta, foram selecionados os 2 (dois) problemas mais similares ao problema consultado. Na Tabela 5.4, são apresentados os resultados obtidos nessa consulta. Os pesos dos atributos utilizados para esta consulta estão descritos na Tabela 4.3.

Estes resultados indicam que os dois problemas recuperados apresentam características semelhantes utilizadas para a resolução do problema, em relação ao problema consultado, uma vez que ambos necessitam realizar múltiplas divisões para sua solução, assim como ocorre na identificação de um número primo.

Tabela 5.4: Resultados obtidos na consulta ao RBC.

| Enunciado | Comp. | Categoria | Assuntos | Padrões | Sim. |
|---|---|---|---|---|---|
| Escreva um programa em PASCAL para ler um número inteiro qualquer e determinar todos os seus divisores exatos. | Fácil | Matemático | Operadores, variáveis, comandos de entrada e saída, comandos de seleção e comandos de repetição. | Repetição contada | 0.72 |
| Escreva um programa em PASCAL para determinar um número inteiro N tal que N + 3N + 5 seja divisível por 121. | Fácil | Matemático | Operadores, variáveis, comandos de entrada e saída, comandos de seleção e comandos de repetição. | Repetição por sentinela | 0.70 |

### Etapa 2

Nesta nova etapa, com a realização das modificações discutidas na Etapa 1, um novo ciclo de experimento foi executado. Com a utilização de uma função de pré-processamento para o atributo enunciado do problema, pode-se observar uma melhoria significativa, conforme ilustrado na Tabela 5.5.

Tabela 5.5: Melhores resultados obtidos com o ajuste de pesos na Etapa 2.

| Enunciado | Assuntos | Padrões | Categoria | Complex. | Média | Desvio Padrão | C.V. (%) |
|---|---|---|---|---|---|---|---|
| 1 | 5 | 1 | 1 | 1 | 0,74 | 0,09 | 12,55 |
| 1 | 5 | 2 | 1 | 1 | 0,72 | 0,11 | 15,55 |
| 1 | 4 | 1 | 1 | 1 | 0,71 | 0,10 | 13,66 |
| 1 | 4 | 2 | 1 | 1 | 0,70 | 0,12 | 17,16 |
| 2 | 5 | 1 | 1 | 1 | 0,69 | 0,11 | 15,37 |

É possível observar, a partir da Tabela 5.5, que com o uso do pré-processamento ocorreu uma redução no valor do coeficiente de variação de 10% em relação aos dados apresentados na Tabela 5.2. Outra consequência importante refere-se ao nível médio de similaridade, que superou os 70%.

Entretanto, analisando separadamente o resultado da similaridade dos atributos, foi observado que a função de similaridade local dos atributos assuntos tratados e padrões de programação não era adequada. Nas Etapas 1 e 2, os valores dos atributos eram representados como uma cadeia de caracteres como, por exemplo, "seleção simples-repetição contada-maior valor". Porém, devido à forma como o cálculo dessa função foi efetuado, uma simples mudança de ordem ou a adição ou supressão de algum valor afeta diretamente o resultado da similaridade.

Assim, supondo que a consulta seja representada pela cadeia de caracteres C = "seleção simples|repetição contada|maior valor"e a função *MaxString()* calcula a similaridade entre a cadeia C e as cadeias M = "seleção simples|maior valor"e N = "seleção simples|repetição contada|menor valor|par|primo", os quais representam os padrões para 2 (dois) problemas na memória de casos, a função *MaxString()* indicaria que o problema N é muito mais similar ao consultado do que o problema M. Isto ocorre porque a maior *substring* da cadeia M em relação a C possui 25 caracteres, enquanto a N possui 32. No entanto, esse resultado não representa a realidade, visto que ambos possuem 2 (dois) padrões idênticos ao problema consultado. Além disso, o problema N apresenta muitos padrões que não existem em C, fato que deveria tornar o problema N menos similar do que o M. Para sanar tal dificuldade, os atributos foram modelados como um conjunto de elementos e novas funções de similaridade foram avaliadas.

Situação semelhante à descrita acima ocorria com a utilização da função *MaxString()* para cálculo da similaridade do atributo enunciado do problema.

### Etapa 3

Para atender às novas alterações, sugeridas na Etapa 2, foram adicionadas à avaliação 4 (quatro) novas funções de similaridade local. Nas Tabelas 5.6 e 5.7 estão descritas a configuração das funções de similaridade e os resultados obtidos nessa etapa, respectivamente.

Tabela 5.6: Funções de similaridade local avaliadas na Etapa 3 do experimento.

| Função | Enunciado | Assuntos da disciplina | Padrões | Complexidade | Categoria |
|---|---|---|---|---|---|
| *Cosine Similarity* | X | X | X | | |
| *Dice's Coefficient* | X | X | X | | |
| *Overlap Coefficient* | X | X | X | | |
| *Jaccard Coefficient* | X | X | X | | |
| *Equal* | | | | X | X |

Com os resultados da Etapa 3, pode-se perceber que, mesmo modificando as funções de similaridade local, os atributos assuntos tratados na disciplina e padrões de programação são de fundamental importância, visto que os melhores resultados obtidos no ajuste de pesos foram aqueles em que esses atributos possuíam pesos maiores do que os demais. Além disso, é possível perceber que, independente da função utilizada, na prática, as combinações de pesos não se alteram.

Tabela 5.7: Melhores resultados obtidos com o ajuste de pesos na Etapa 3.

| Função | Enun. | Assuntos | Padrões | Categ. | Complex. | Média | D.V. | C.V. (%) |
|---|---|---|---|---|---|---|---|---|
| *Cosine Coefficient* | 1 | 5 | 1 | 1 | 1 | 0,75 | 0,09 | 12,25 |
| | 1 | 5 | 2 | 1 | 1 | 0,73 | 0,11 | 15,05 |
| | 1 | 4 | 1 | 1 | 1 | 0,72 | 0,09 | 13,15 |
| | 1 | 5 | 3 | 1 | 1 | 0,72 | 0,13 | 17,99 |
| | 2 | 5 | 1 | 1 | 1 | 0,72 | 0,10 | 14,61 |
| *Jaccard Coefficient* | 1 | 5 | 1 | 1 | 1 | 0,72 | 0,11 | 15,55 |
| | 1 | 5 | 2 | 1 | 1 | 0,70 | 0,12 | 18,23 |
| | 1 | 4 | 1 | 1 | 1 | 0,69 | 0,11 | 16,57 |
| | 1 | 5 | 3 | 1 | 1 | 0,69 | 0,14 | 20,87 |
| | 2 | 5 | 1 | 1 | 1 | 0,68 | 0,12 | 18,39 |
| *Dice's Coefficient* | 1 | 5 | 1 | 1 | 1 | 0,75 | 0,09 | 12,61 |
| | 1 | 5 | 2 | 1 | 1 | 0,73 | 0,11 | 15,42 |
| | 1 | 4 | 1 | 1 | 1 | 0,72 | 0,09 | 13,52 |
| | 1 | 5 | 1 | 1 | 1 | 0,72 | 0,10 | 14,99 |
| | 2 | 5 | 3 | 1 | 1 | 0,72 | 0,13 | 18,37 |
| *Overlap Coefficient* | 1 | 5 | 1 | 1 | 1 | 0,79 | 0,07 | 09,26 |
| | 1 | 5 | 2 | 1 | 1 | 0,77 | 0,09 | 12,59 |
| | 1 | 4 | 1 | 1 | 1 | 0,77 | 0,12 | 16,06 |
| | 1 | 5 | 1 | 1 | 1 | 0,76 | 0,14 | 19,25 |
| | 2 | 5 | 3 | 1 | 1 | 0,76 | 0,08 | 11,14 |

Das novas funções avaliadas, a *Overlap Coefficient* se destacou com as maiores médias de similaridade – com valores entre 75% e 80% – e com os menores desvios padrão. Porém, diante da análise subjetiva, concluiu-se que os melhores resultados foram os obtidos com a função *Cosine Coefficient*, muito embora a função *Dice's Coefficient* tenha apresentado também resultados bastante promissores.

Para exemplificar os resultados obtidos com a utilização da função *Cosine Coefficient*, ao final dessa etapa de avaliação, na Tabela 5.8, está apresentada a descrição de um problema que foi consultado.

Tabela 5.8: Descrição do problema consultado no RBC.

| Enunciado | Comp. | Categoria | Assuntos | Padrões |
|---|---|---|---|---|
| Escreva um programa que leia um conjunto de palavras e um parágrafo, em seguida, remova do parágrafo as palavras informadas e imprima a frequencia de cada palavra restante que se repete no parágrafo. | Médio | SI | Comandos de repetição e String | Seleção simples e Repetição com Sentinela |

A partir dessa consulta, foram selecionados os 3 (três) problemas mais similares ao pro-

blema consultado. Na Tabela 5.9, são apresentados os resultados obtidos nessa consulta. Os pesos dos atributos para esta consulta estão descritos na Tabela 4.3.

Tabela 5.9: Resultados obtidos na consulta ao RBC.

| **Enunciado** | **Comp.** | **Categoria** | **Assuntos** | **Padrões** | **Sim.** |
|---|---|---|---|---|---|
| Escreva um programa que retire as stopwords do parágrafo informado e, em seguida, apresente a frequência com que cada palavra restante se repete no parágrafo. Algumas stopwords são a, e, o, da, de, do, para, um e uma. | Médio | SI | Comandos de Repetição e String | Seleção simples e Repetição com Sentinela | 0.85 |
| Escreva um professor de português precisa de um programa que dado um parágrafo, apresente a frequência com que cada palavra se repete neste parágrafo. | Médio | SI | String | Repetição por sentinela | 0.65 |
| Escreva a função contacaracteres que receba uma string e retorne um dicionário representando a contagem de caracteres contidos na string. O dicionário resultante deve conter exclusivamente os caracteres presentes na string. | Médio | SI | String | Repetição por sentinela | 0.60 |

Os problemas recomendados pelo sistema de raciocínio baseado em casos apresentam características semelhantes em relação ao consultado, visto que os 3 (três) problemas necessitam, no mínimo, da criação de uma rotina que calcule a frequência das palavras.

## 5.2 Avaliação do ambiente *Analogus*

Com o objetivo de avaliar o *Analogus* como um ambiente capaz de auxiliar o aluno na resolução de problemas de programação por meio do raciocínio por analogia, foi realizado um ensaio experimental com 10 alunos do curso de Ciência da Computação da Universidade Federal de Campina Grande.

Para avaliação, foram criados 3 (três) documentos, sendo esses:

- **Questionário do perfil do aluno:** questionário investigativo com o objetivo de identificar o perfil do aluno que participou do experimento;
- **Roteiro de atividades:** roteiro que descreve as atividades a serem realizadas pelo aluno na utilização do ambiente;

- **Questionário de avaliação do ambiente:** questionário que investiga o nível de satisfação do aluno em relação ao auxílio fornecido pelo ambiente na resolução dos problemas de programação por meio do raciocínio por analogia.

Neste experimento, os participantes inicialmente preencheram o questionário do perfil do aluno, informando seus dados pessoais e descrevendo a estratégia por eles empregada na resolução dos problemas de programação. Em seguida, foi realizado um treinamento sobre o ensino de programação por meio de padrões, no qual eram apresentados os padrões disponíveis na ferramenta e informado como os alunos poderiam utilizá-los para resolver os problemas. Após o treinamento, os participantes da avaliação acessavam o ambiente *Analogus* e executavam o roteiro de atividades. Uma vez finalizado o roteiro, eles preenchiam o questionário de avaliação do ambiente.

Na avaliação, cada participante utilizou o ambiente *Analogus* para solucionar 2 (dois) problemas de programação distintos, sendo esses:

- Escreva um programa que soma dois vetores inteiros, de mesmo tamanho, gerados aleatoriamente e imprime o maior e menor inteiro da soma, indicando se tais números são par ou ímpar. Não use as funções *min* ou *max* de *python*.

- Escreva um programa que retire as *stopwords* do parágrafo informado e, em seguida, apresente a freqüência com que cada palavra restante se repete no parágrafo. Algumas *stopwords* são a, e, o, da, de, do, para, um e uma.

Para o primeiro problema, o roteiro de atividades define cada passo a ser executado pelo participante até o instante que ele inicia a codificação da solução. Para o segundo problema, o participante fica livre para manipular o ambiente, sem a necessidade de seguir um roteiro fixo. No Apêndice C estão apresentados os questinários do perfil do aluno, da avaliação do ambiente e o do roteiro de atividades.

A memória de casos do *Analogus*, para essa avaliação, era constituída por 10 (dez) problemas de programação distintos, previamente solucionados pelos alunos em sala de aula, durante a disciplina de programação.

### 5.2.1 Análise do Perfil dos Participantes

Os participantes da avaliação foram 10 (dez) alunos, de ambos os sexos, regularmente matriculados no 2° semestre do curso de Ciência da Computação da Universidade Federal de Campina Grande, com idade média de 19 anos e experiência com programação de 6 (seis) meses a 1 (um) ano. Embora a quantidade de alunos tenha sido limitada, é importante ressaltar que esse valor representa cerca de 30% dos alunos de uma turma de introdução à programação e que cada aluno necessitou de um tempo médio de 1 (uma) hora e 30 (trinta) minutos para executar todo o experimento.

Diante das respostas dos participantes para o questionário do perfil do aluno, observou-se que 100% deles gostam da atividade de programar e 80% acreditam que está é uma atividade classificada como "nem fácil nem difícil"de ser realizada.

Dos questionados, 40% informaram que a maior dificuldade no momento em que estão programando é a criação de uma solução para o problema, seguida pelas dificuldades em entender o problema, utilizar a sintaxe da linguagem de programação e depurar a solução, cada uma delas com 20%, conforme ilustrado na Figura 5.1. Este resultado está de acordo com a literatura que aponta a criação de uma solução como uma das fases que os alunos sentem maior dificuldade.

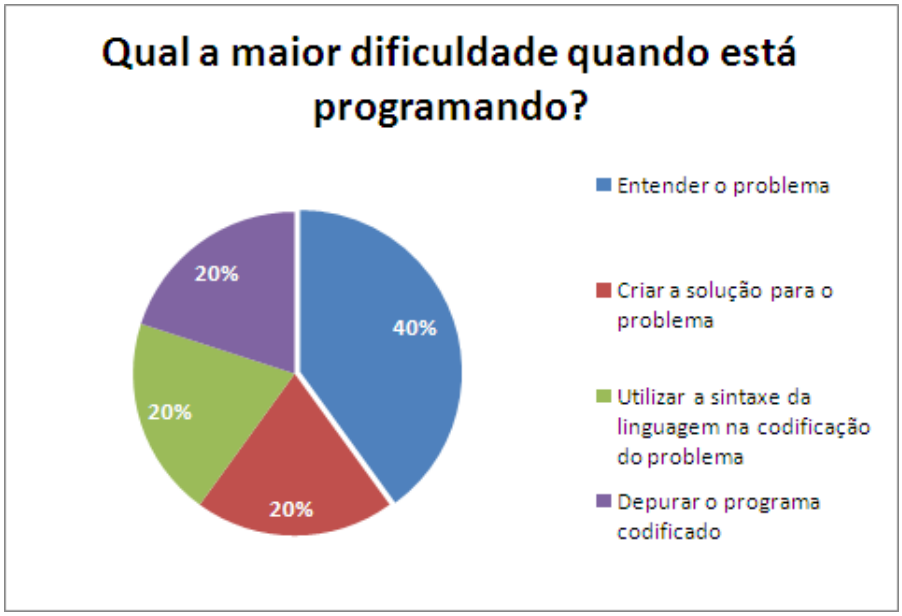



Figura 5.1: Qual sua maior dificuldade encontrada quando está programando?

Vale observar, a partir da Figura 5.2 que 90% dos entrevistados afirmam que se recordam

de soluções semelhantes para solucionar um novo problema. Esse número comprova que os alunos iniciantes utilizam essa estratégia de raciocínio – po analogia – ou tentam utilizar, uma vez que é natural aos seres humanos.

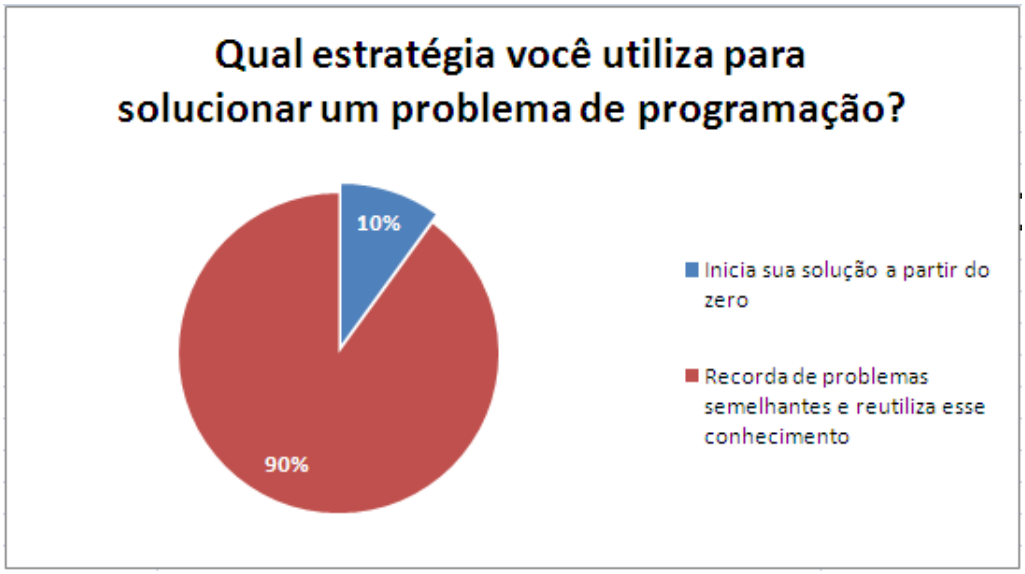



Figura 5.2: Qual estratégia você utiliza para solucionar um problema de programação?

Outro fato relevante reside no fato de que 60% dos participantes informaram que aprenderam a programar por meio de padrões. Contudo, quando questionados sobre essa resposta, eles foram informados de que nas aulas os professores ensinavam a programar de maneira semelhante à apresentada no treinamento inicial. Porém, sem a formalização de que as estruturas apresentadas eram padrões.

### 5.2.2 Análise da Avaliação do Ambiente

Os participantes da avaliação, após executarem o roteiro (*vide* Apêndice C) no *Analogus*, preencheram o questionário de avaliação do ambiente, o qual consistia em um conjunto de afirmações que o aluno poderia: concordar plenamente, concordar, nem concordar nem discordar, discordar ou discordar plenamente.

Para cada afirmação, o participante deveria informar também o grau de confiabilidade da sua resposta, ou seja, indicar qual o seu nível de segurança ao escolher cada afirmação (em uma escala de 0 a 10, em que o último indica 100% de confiabilidade). Nesta seção, as respostas dos participantes da avaliação serão analisadas estatisticamente.

Quando mencionado aos participantes que "Recordar problemas semelhantes previa-

mente solucionados é uma estratégia interessante para resolver novos."50% concordaram plenamente com a afirmação, com uma média de confiabilidade (M.C.) de 9,5 em sua resposta. Os demais participantes concordaram com a afirmação, com uma confiabilidade média de 9,25. Com isso, é possível perceber que os alunos reconhecem a importância do raciocínio por analogia na resolução de problemas de programação. Na Figura 5.3, estão sumarizados esses resultados.

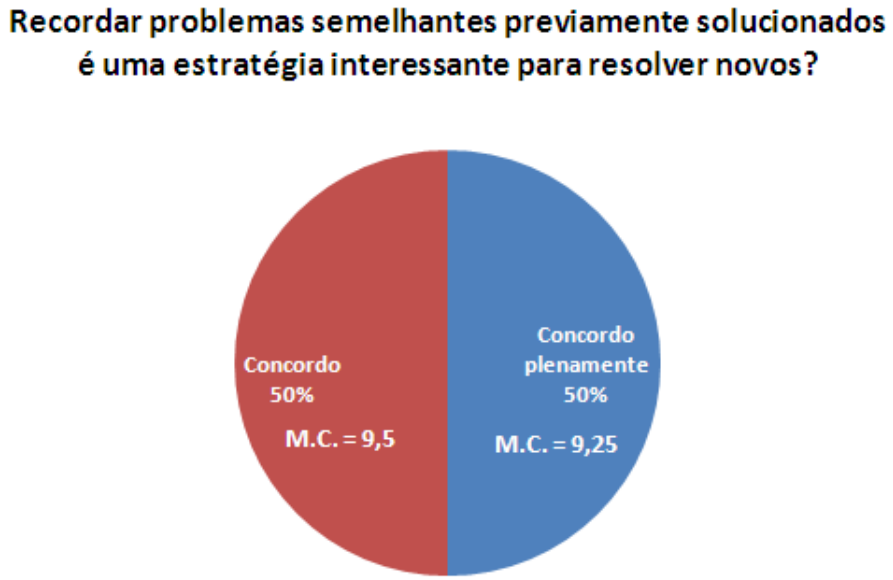



Figura 5.3: Raciocínio por analogias como estratégia de resoluções de problemas.

Com relação ao ambiente, após utilizá-lo, 60% dos participantes concordaram plenamente (conforme Figura 5.4), com uma confiabilidade média de 9,15, que o *Analogus* auxilia o aluno a lembrar de problemas previamente solucionados e similares ao atual. Além disso, todos os participantes confirmaram que utilizaram o conhecimento das soluções dos problemas similares indicados pelo ambiente na resolução do problema atual e atribuíram a nota média de 8,4 para semelhança entre esse problema e o indicado pelo *Analogus*, em uma escala de 0 a 10.

Concernente aos aspectos que auxiliaram na identificação dos problemas similares, 80% dos participantes concordaram que os padrões de programação ajudaram nesse processo, de acordo com a Figura 5.5. Tal concordância comprova a importância da utilização dos padrões na representação do caso, uma vez que o próprio aluno utiliza a informação para verificar a similaridades entre os problemas.

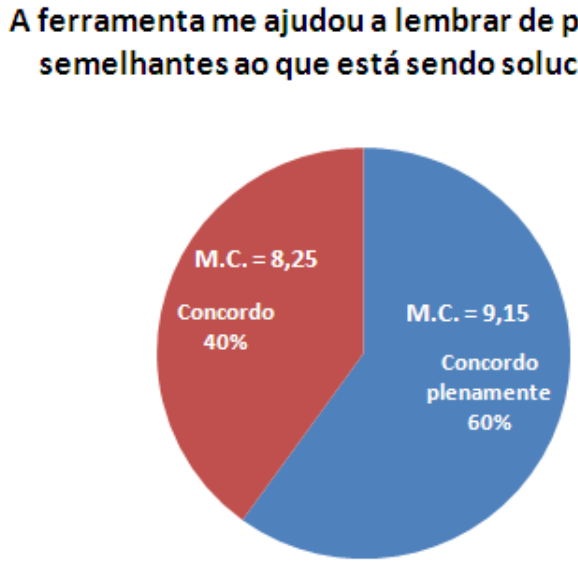



Figura 5.4: Indicação de problemas similares pelo *Analogus*.

Além disso, é importante ressaltar que os alunos participantes do experimento não cursaram a disciplina de programação com foco na utilização de padrões no ensino de programação, apenas participaram de um breve treinamento efetuado antes do experimento.

Assim, acredita-se que resultados ainda melhores podem ser obtidos com a utilização dessa ferramenta em cursos que sigam a metodologia de ensino de programação por padrões, principalmente porque 100% dos entrevistados afirmaram com uma confiabilidade média de 9,2, que a ferramenta auxilia na utilização de tais padrões.

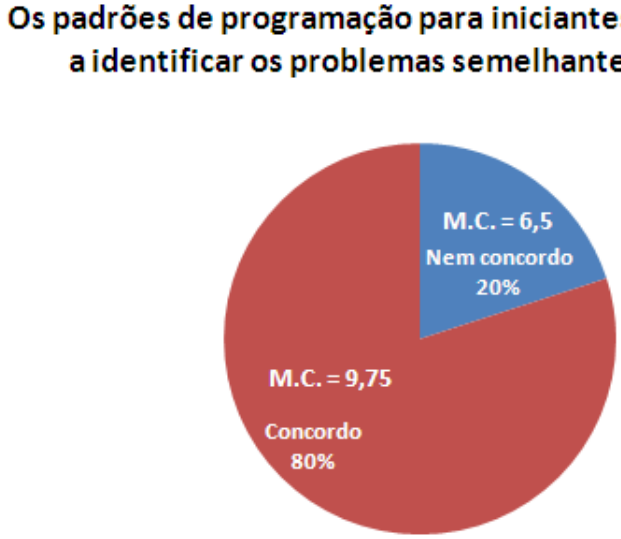



Figura 5.5: Os padrões ajudaram na identificação dos problemas.

Analisando as respostas dos alunos participantes sobre os questionamentos da utilização do professor virtual no ambiente, concluiu-se que 90% concordaram que o agente inteligente de diálogo ajudou na identificação de problemas semelhantes ao atual. Além disso, 100% concordaram que o agente estimula essa habilidade e a reflexão sobre os aspectos que tornam os problemas semelhantes.

Assim, é possível verificar que esses dados comprovam que, a partir do diálogo, é possível auxiliar o aluno no aprimoramento da habilidade de identificação de problemas similares, mostrando quais são as semelhanças e diferenças entre os problemas, ou como identificar um problema similar.

É importante destacar também que 90% dos participantes concordaram que a *interface* do ambiente é clara e compreensível, conforme ilustrado na Figura 5.6.

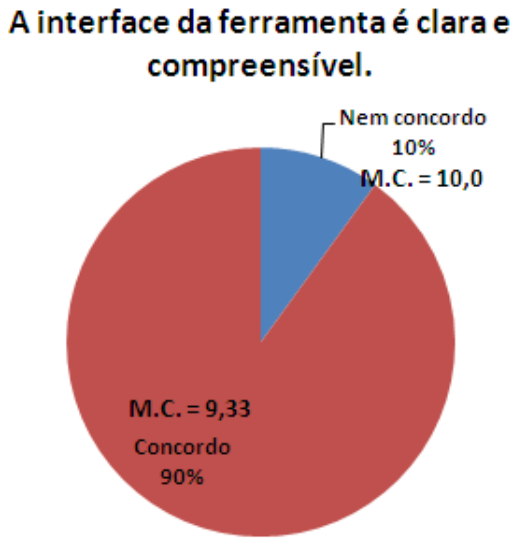



Figura 5.6: *Interface* clara e compreensível.

Os participantes concordaram, unanimente, que o ambiente auxilia na resolução de problemas de programação por meio do raciocínio por analogia, que ficaram satisfeitos com o *Analogus* e que recomendariam, sem hesitar, o uso do ambiente no ensino das disciplinas de introdução à programação.

## 5.3 Discussão

A avaliação da recuperação dos problemas de programação, bem como o *feedback* dos alunos acerca dos aspectos de similaridade entre os problemas, comprovaram que a representação do caso foi adequada para o domínio.

Além disso, o ensaio experimental, realizado com os alunos, mostra indícios significativos de que o *Analogus* é capaz de apoiar o aluno na resolução de problemas de programação, por meio de analogia. Isso porque, os participantes confirmaram que o ambiente apontou corretamente problemas úteis para resolução do problema atual.

Entretanto, acredita-se que, para mensurar resultados mais concludentes, é necessário realizar um experimento mais criterioso, aplicando o ambiente durante alguns sementres em diferentes turmas.

# Capítulo 6

# Considerações Finais e Trabalhos Futuros

Neste capítulo são descritas as considerações finais da pesquisa, bem como, são apresentadas sugestões para trabalhos futuros nas Seções 6.1 e 6.2, respectivamente.

## 6.1 Considerações Finais

O aprimoramento da habilidade de resolver problemas de programação por meio do raciocínio por analogia pode ser comprovado pelos resultados da avaliação do ambiente *Analogus*, conforme discutido no Capítulo 5.

No desenvolvimento do *Analogus* foi realizada a integração de um sistema de raciocínio baseado em casos com um agente inteligente de diálogo, no intuito de fornecer ao aluno um ambiente de resolução de problemas de programação capaz de auxiliá-lo na resolução por meio de analogias. Para tanto, foi necessário desenvolver um módulo de resolução de problemas, o qual controla e integra as ferramentas citadas.

A utilização de ferramentas de terceiros - *jColibi* e *Chatterbean* - facilitaram o desenvolvimento e possibilitaram a instanciação do sistema de raciocínio baseado em casos e do agente inteligente de diálogo, de forma ágil, a fim de verificar se a solução proposta era capaz de atingir os objetivos deste trabalho.

Diante do exposto, tornou-se possível analisar formas adequadas de representar e recuperar o caso no domínio de auxílio ao ensino de programação e verificar se o diálogo era adequado para estimular a habilidade dos alunos em identificar similaridades entre os problemas.

A escolha das funções de similaridade global e locais foi de fundamental importância, dado que estas tiveram o objetivo de verificar a melhor função para cada índice representado no caso.

Conforme a avaliação dos resultados, foi possível observar uma forte indicação de eficiência da metodologia utilizada para representação e recuperação do caso, uma vez que os participantes da avaliação informaram que os problemas recomendados foram úteis para solucionar um novo problema apresentado.

Com os resultados obtidos, também foi possível observar que os índices utilizados na recuperação do caso são os mesmos empregados pelos alunos na identificação dos problemas similares no momento da avaliação, mostrando que, com essa forma de representação, o raciocínio por analogia empregado pela ferramenta está em sintonia com o raciocínio dos alunos.

Vale destacar que, a utilização de padrões de programação para alunos iniciantes favoreceu o raciocínio por meio de analogia para resolução de problemas, principalmente porque tais padrões seguem, desde a sua concepção, a ideia de reutilização do conhecimento comum entre situações.

A inclusão desses padrões na modelagem do caso facilitou, portanto, a identificação de problemas similares, tanto pelos alunos como pelo sistema de raciocínio baseado em casos, além de possibilitar a utilização dessa ferramenta em alunas práticas das diciplinas de introdução à programação, de cursos que seguem a metodologia de ensino de programação por meio de padrões.

A utilização do agente inteligente de diálogo possibilitou uma melhor interação com o aluno, fazendo com este refletisse sobre os aspectos de similaridade entre os problemas. Além disso, as expressões faciais do agente motivaram o aluno no diálogo, fato que foi observado durante a avaliação.

É importante ressaltar outra característica fundamental do professor virtual: a disponibilidade. Ou seja, o *Analogus* está disponibilizado como uma aplicação *Web*, permitindo que aluno elucide as suas dúvidas com o professor virtual ou solucione um novo problema no horário que lhe é mais adequado.

Embora a base de conhecimento do agente ainda seja bastante limitada, os resultados obtidos indicam que a sua integração com o raciocínio baseado em casos favorece a resolução

do problema.

Apesar de a avaliação ter sido realizada apenas com 10 (dez) participantes, é importante destacar que esse número representa cerca de 30% da quantidade de alunos matriculados em turmas de introduação à programação. Destaca-se também que, para a realização do roteiro do experimento, cada participante necessitou de, em média, 1 (uma) hora e 30 (trinta) minutos. Este foi um fator que também implicou na redução do número de participantes, uma vez que cada um deles tinha que ser acompanhado durante todo o processo de avaliação, a fim de melhor observar as facilidades e dificuldades apresentadas pelo aluno na utilização da ferramenta.

É importante destacar, que os resultados obtidos neste estudo foram publicados em conferências nacionais importantes da área de informática em educação, conforme o Apêndice C.4

Diante do exposto, pode-se concluir que a utilização conjunta dessas duas técnicas é capaz de auxliar no aprimoramento da habilidade do aluno em resolver problemas de programação por meio de analogia, principalmente, com a utilização diária da ferramenta na resolução das listas de exercícios de programação fornecidas pelo professor.

## 6.2 Sugestões para Trabalhos Futuros

Julga-se fundamental, como trabalho futuro, efetuar um experimento mais rigoroso para obtenção de resultados mais concludentes sobre o nível do auxílio fornecido pelo ambiente na resolução de problemas de programação por meio de raciocínio por analogia. Esta nova avaliação deve ser efetuada ao longo de, no mínimo, 3 (três) semestres da disciplina de introdução à programação. Para tanto, é indicado que o curso adote a metodologia de ensino programação por meio de padrões. Os alunos das turmas avaliadas deverão ser separados em 2 (dois) grupos: o grupo experimental e o de controle. Neste novo experimento, os alunos aprenderiam nas aulas teóricas os padrões e problemas exemplificados pelo professor e nas aulas práticas usariam o *Analogus* para solucionar os problemas sugeridos pelo professor. Desse modo, é possível avaliar o ambiente no cenário ideal, no qual o *Analogus* está associado a uma metodologia de ensino por meio de padrões.

É preciso, na visão do aluno, melhorar alguns aspectos na *interface* do *Analogus* conforme sugestões apresentadas pelos participantes da avaliação, destacando-se: emitir alertas

sonoros ou visuais para chamar a atenção do aluno quando o professor virtual estiver falando; destacar os trechos de código das soluções passadas que serão úteis para solucionar o problema atual; não permitir a codificação no momento em que o aluno estiver dialogando com o professor virtual, evitando a desconcentração no diálogo.

O diálogo do professor virtual ainda pode ser enriquecido com a utilização de ferramentas de recuperação de informação na *Web*, de tal forma que dúvidas sobre a sintaxe ou funções específicas da linguagem possam ser buscadas na internet, quando a base de conhecimento não possuir informações necessárias para satisfazer as dúvidas do aluno.

Nesse sentido, seria importante também adicionar um módulo de *feedback* do aluno. Esse módulo permitiria ao aluno informar se o diálogo com o professor virtual foi suficiente para auxiliá-lo na resolução do problema. A partir da análise do *feedback*, o diálogo poderia ser melhorado e, com o tempo, atender plenamente aos questionamentos do aluno.

Outra característica importante que poderia ser adicionada ao ambiente seria a criação de um módulo de visualização e edição gráfica dos padrões de programação, permitindo que o aluno modele a solução com os blocos de padrões e, em seguida, instancie para a linguagem aprendida. Para tanto, os padrões de programação devem ser mapeados em figuras ou blocos e o ambiente deve permitir o sequenciamento ou aninhamento desses para criação da solução do problema. Essa característica poderá favorecer o raciocínio por meio de analogia para a resolução de problemas de programação, uma vez que algumas pessoas sentem facilidade em recordar imagens visualizadas anteriormente e associá-las com outras semelhantes.

O *Analogus* se limita à experiência do próprio aluno, ou seja, o raciocinador baseado em casos recupera e recomenda apenas problemas que o aluno tenha solucionado. Porém, em algumas situações, a memória do aluno ainda não possui informações relevantes para solucionar tal problema, sendo necessário obter experiências externas, seja buscando na internet, seja discutindo com outros alunos.

Assim, adicionar essa característica de compartilhamento de experiências entre alunos e agentes computacionais, uma vez que alunos de uma mesma sala possuem uma memória de casos bastante parecida, proporcionaria uma maior diversidade de experiências, favorecendo o raciocínio por meio de analogias. Contudo, para tanto, é necessário investigar como deve ocorrer a colaboração e a cooperação entre os agentes humanos e computacionais, como os grupos devem ser formados a fim de maximizar a troca de experiências e como adicionar na

memória de casos do aluno essa experiência externa garantindo sua autoria.

# Bibliografia


[AP94] Agnar Aamodt and Enric Plaza. Case-based reasoning: foundational issues, methodological variations, and system approaches. *AI Communications*, 7(1):39–59, Março 1994.

[BBD+01] Joseph Bergin, Alyce Brady, Robert Duvall, Viera Proulx, and Richard Rasala. Using patterns in the classroom. In *CCSC '01: Sixth annual CCSC northeastern conference on The journal of computing in small colleges*, pages 5–7, USA, 2001. Consortium for Computing Sciences in Colleges.

[BD06] Leliane N. Barros and Karina V. Delgado. Aprendizado de programação. In *XIV Workshop sobre Educação em Computação*. Anais do XIV Workshop sobre Educação em Computação, 2006.

[BDM04] L. N. Barros, K. V. Delgado, and A. Machion. An its for programming to explore practical reasoning. In *Simpósio Brasileiro de Informática na Educação*, pages 207–216. Editora SBC, Novembro 2004.

[BFP03] Stephen Bloch, Kathi Fisler, and Viera K. Proulx. Introductory computer science with focus on program design. *J. Comput. Small Coll.*, 18(5):70–71, 2003.

[BKP+99] Joseph Bergin, Amruth Kumar, Viera K. Proulx, Myles Mcnally, Alyce F. Brady, David Mutchler, Stephen Hartley, Richard Rasala, Charles Kelemen, Rocky Ross, and Frank Klassner. Resources for next generation introductory cs courses: Report of the iticse'99 working group on resources for the next generation cs 1 course. In *ITiCSE-WGR '99: Working group reports from ITiCSE on Innovation and technology in computer science education*, pages 101–105, New York, NY, USA, 1999. ACM.

[BRaN+08a] E. Barreto, M. Romão, F. Neto, A. Mendonça, G. P. Santos Júnior, and J. M. Fechine. Composição dinâmica de listas de problemas de programação utilizando algoritmos genéticos interativos. volume 32, pages 229–236. Revista Hífen, 2008.

[BRaN+08b] E. Barreto, M. Romão, F. Neto, A. Mendonça, G. P. Santos Júnior, and J. M. Fechine. Uma aplicação de algoritmos genéticos interativos para composição de listas de problemas de programação. In *XIX Simpósio Brasileiro de Informática na Educação*. Anais do XIX Simpósio Brasileiro de Informática na Educação, 2008.

[CBR03] *Applying Knowledge Management: Techniques for Building Corporate Memories (The Morgan Kaufmann Series in Artificial Intelligence)*. Morgan Kaufmann, 1 edition, Janeiro 2003.

[CL99] Michael J. Clancy and Marcia C. Linn. Patterns and pedagogy. *SIGCSE Bull.*, 31(1):37–42, 1999.

[dBADM05] Leliane N. de Barros, Ana, Karina V. Delgado, and Patricia M. Matsumoto. A tool for programming learning with pedagogical patterns. In *eclipse '05: Anais do 2005 OOPSLA workshop on Eclipse technology eXchange*, pages 125–129, New York, NY, USA, 2005. ACM.

[Del05] Karina V. Delgado. Diagnóstico baseado em modelos num sistema tutor inteligente para programação com padrões pedagógicos. Master's thesis, Instituto de Matemática e Estatística da Universidade de São Paulo, 2005.

[dMMB+05] Ramon de Mantaras, David Mcsherry, Derek Bridge, David Leake, Barry Smyth, Susan Craw, Boi Faltings, Mary L. Maher, Michael T. Cox, Kenneth Forbus, Mark Keane, Agnar Aamodt, and Ian Watson. Retrieval, reuse, revision and retention in case-based reasoning. *The Knowledge Engineering Review*, 20(03):215–240, 2005.

[Gar08] Juan A. García. *jCOLIBRI: A multi-level platform for building and generat-*

*ing CBR systems*. PhD thesis, Universidad Complutense de Madrid, Outubro 2008.

[GHC+01] D. Ginat, B. Hamberman, D. Cohen, D. Catz, and O. Muller. *Pattern in Computer Science*. 2001.

[GHJV95] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns*. Addison-Wesley Professional, Janeiro 1995.

[GLT03] Dedre Gentner, Jeffrey Loewenstein, and Leigh Thompson. Learning and transfer: A general role for analogical encoding. *Journal of Educational Psychology*, 95:393–408, 2003.

[Gly07] Shawn Glynn. The teaching with analogies. *Science and Children*, 44:52–55, 2007.

[Gos93] Usha Goswami. *Analogical Reasoning In Children (Essays in Developmental Psychology)*. Psychology Press, 1 edition, Agosto 1993.

[GPjBS06] Linda Grandell, Mia Peltomäki, Ralph johan Back, and Tapio Salakoski. Why complicate things? introducing programming in high school using python. pages 71–80, Hobart, Tasmania, Australia, 2006. Eighth Australasian Computing Education Conference (ACE2006).

[HM08] B. Haberman and O. Muller. Teaching abstraction to novices: Pattern-based and adt-based problem-solving processes. In *38th ASEE/IEEE Frontiers in Education Conference*, Outubro 2008.

[JAAP05] Juan, Antonio, Belen D. Agudo, and Pedro. jcolibri 1.0 in a nutshell. a software tool for designing cbr systems. In M. Petridis, editor, *Proccedings of the 10th UK Workshop on Case Based Reasoning*, pages 20–28. CMS Press,University of Greenwich, 2005.

[KCGC05] Janet L. Kolodner, Michael T. Cox, and Pedro A. Gonzalez-Calero. Case-based reasoning-inspired approaches to education. *The Knowledge Engineering Review*, 20(03):299–303, 2005.

[KF07] Helton M. Kraus and Anita M. Fernandes. Desenvolvimento de um chatterbot para Área imobiliária integrando raciocínio baseado em casos. volume 31, pages 37–43. Revista Hífen, 2007.

[Kol93] Janet Kolodner. *Case-Based Reasoning (Morgan Kaufmann Series in Representation & Reasoning)*. Morgan Kaufmann, Setembro 1993.

[Kos99] Marco A. Koslosky. Aprendizagem baseada em casos - um ambiente de ensino de programação. Master's thesis, Universidade Federal de Santa Catarina, 1999.

[LCDT03] M. D. Leonhardt, Daiane D. Castro, Renato L. Dutra, and L. M. R. Tarouco. Elektra: Um chatterbot para uso em ambiente educacional. volume 1. Renote - Revista Novas Tecnologias na Educação, 2003.

[Leo05] Michelle D. Leonhardt. Doroty: um chatterbot para treinamento de profissionais atuantes no gerenciamento de redes de computadores. Master's thesis, Universidade Federal do Rio Grande do Sul, 2005.

[LNT03] Michelle D. Leonhardt, Ricardo Neisse, and Liane M. Tarouco. Meara: Um chatterbot temático para uso em ambiente educacional. In *XIV Simpósio Brasileiro de Informática na Educação*, pages 85–92, 2003.

[LT06] Michelle D. Leonhardt and Liane M. Tarouco. Doroty: um chatterbot para aprendizado baseado em problemas aplicado ao treinamento de profissionais de gerência de redes. In *XVII Simpósio Brasileiro de Informática na Educação*, 2006.

[LTV+07] M. D. Leonhardt, L. Tarouco, R. M. Vicari, E. R. Santos, and Dos. Using chatbots for network management training through problem-based oriented education. In *Advanced Learning Technologies, 2007. ICALT 2007. Seventh IEEE International Conference on*, pages 845–847, 2007.

[Mat00] Mauro M. Mattos. Construção de abstrações em lógica de programação. In *XX Congresso Nacional da Sociedade Brasileira de Computação*, volume 1, pages 60+. Editora Universitária Champagnat, 2000.

[Mat02] Mauro M. Mattos. Learning how to build abstractions in programming logics classes. In *Congresso Ibero-Americano de Informática*, volume 6, 2002.

[Men08] Andréa P. Mendonça. Entendimento de problemas: Uma investigação exploratória com alunos iniciantes de programação. Technical report, Departamento de Sistemas e Computação. Coordenação de Pós-Graduação em Ciência da Computação, 2008.

[MFL99] Mauro M. Mattos, Andrino Fernandes, and Oscar C. López. Sistema especialista para apoio ao aprendizado de lógica de programação. In *VII Congresso Iberoamericano de Educación Superior en Computación.- CIESC99*, 1999.

[MGH07] Orna Muller, David Ginat, and Bruria Haberman. Pattern-oriented instruction and its influence on problem decomposition and solution construction. In *ITiCSE '07: Anais do 12th annual SIGCSE conference on Innovation and technology in computer science education*, pages 151–155, New York, NY, USA, 2007. ACM.

[MH05] Orna Muller and Bruria Haberman. Guidelines for a multiple-goal cs introductory course: algorithmic problem-solving woven into oop. In *ITiCSE '05: Anais do 10th annual SIGCSE conference on Innovation and technology in computer science education*, page 356, New York, NY, USA, 2005. ACM.

[MHA04] Orna Muller, Bruria Haberman, and Haim Averbuch. (an almost) pedagogical pattern for pattern-based problem-solving instruction. In *ITiCSE '04: Anais do 9th annual SIGCSE conference on Innovation and technology in computer science education*, pages 102–106, New York, NY, USA, 2004. ACM.

[Mul05] Orna Muller. Pattern oriented instruction and the enhancement of analogical reasoning. In *ICER '05: Anais do 2005 international workshop on Computing education research*, pages 57–67, New York, NY, USA, 2005. ACM.

[Mul08] O. Muller. Developing algorithmic problem-solvin skills - the rationale underlying a course. Outubro 2008.

[NDJ01] Ronaldo L. Nagem, Dulcinéia, and Jully. Uma proposta de metodologia de ensino com analogias. *Revista Portuguesa de Educação*, 14:197–213, 2001.

[OGP08] C. H. J. S. Oliveira, R. R. Greghi, and E. P. Pimentel. Ambiente para assistência à aprendizagem e programação baseado em padrões pedagógicos. In *XIX Simpósio Brasileiro de Informática em Educação*, 2008.

[Old05] Joseph D. Oldham. What happens after python in cs1? *J. Comput. Small Coll.*, 20(6):7–13, 2005.

[PCR08] R. Pirrone, V. Cannella, and G. Russo. Gaiml: A new language for verbal and graphical interaction in chatbots. In *Complex, Intelligent and Software Intensive Systems, 2008. CISIS 2008. International Conference on*, pages 715–720, 2008.

[PM06] Holly Patterson-McNeill. Experience: from c++ to python in 3 easy steps. *J. Comput. Small Coll.*, 22(2):92–96, 2006.

[Pol71] G. Polya. *How to Solve It*. Princeton University Press, Novembro 1971.

[Poz02] J. I. Pozo. *Teorias Cognitivas da Aprendizagem*. 3 edition, 2002.

[Pro00] Viera K. Proulx. Programming patterns and design patterns in the introductory computer science course. In *SIGCSE '00: Anais do thirty-first SIGCSE technical symposium on Computer science education*, pages 80–84, New York, NY, USA, 2000. ACM.

[RA05] MICHAEL M. RICHTER and AGNAR AAMODT. Case-based reasoning foundations. *The Knowledge Engineering Review*, 20(03):203–207, 2005.

[SJCF08] Gilson P. S. Santos Júnior, Evandro B. Costa, and Joseana M. Fechine. Raciocínio baseado em casos para auxílio a alunos na resolução de problemas por analogia - uma abordagem para representação e recuperação de casos. In *XIX Simpósio Brasileiro de Informática em Educação*, pages 593–602, 2008.

[SJFC09] Gilson P. Santos Júnior, Joseana M. Fechine, and Evandro Costa. Analogus: Um ambiente para auxílio ao ensino de programação orientado pelo raciocínio

por analogia. In *Anais do XVII Workshop sobre Educação em Computação*, July 2009.

[Som06] Ian Sommerville. *Software Engineering: (Update) (8th Edition) (International Computer Science Series)*. Addison Wesley, Junho 2006.

[SP04a] Simon Shiu and Sankar K. Pal. *Foundations of Soft Case-Based Reasoning (Wiley Series on Intelligent Systems)*. Wiley-Interscience, 1 edition, Março 2004.

[SP04b] Simon C. K. Shiu and Sankar K. Pal. Case-based reasoning: Concepts, features and soft computing. *Applied Intelligence*, 21(3):233–238, 2004.

[WAB05] ROSINA O. WEBER, KEVIN D. ASHLEY, and STEFANIE BRUNINGHAUS. Textual case-based reasoning. *The Knowledge Engineering Review*, 20(03):255–260, 2005.

[WB01] Gerhard Weber and Peter Brusilovsky. Elm-art: An adaptive versatile system for web-based instruction. *International Journal of Artificial Intelligence in Education*, 12:351–384, 2001.

[WM94] Ian Watson and Farhi Marir. Case-based reasoning: A review. 9(4), 1994.

# Apêndice A

# Lista de *stopwords* e palavras comuns ao domínio

## A.1 Lista de *stopwords*

acerca – agora – algumas – alguns – ali – ambos – antes – apontar – aquela – aquelas – aquele – aqueles – aqui – atrás – bem – bom – cada – caminho – cima – com – como – comprido – conhecido – corrente – das – debaixo – dentro – desde – desligado – deve – devem – deverá – direita – diz – dizer – dos – e – é – ela – ele – eles – em – enquanto – então – está – estão – estado – estar – estará – este – estes – esteve – estive – estivemos – estiveram – eu – fará – faz – fazer – fazia – fez – fim – foi – fora – horas – iniciar – inicio – ir – irá – isto – ligado – maioria – maiorias – mais – mas – mesmo – meu – muito – muitos – nós – não – nome – nosso – novo – o – onde – os – ou – outro – para – parte – pegar – pelo – pessoas – pode – poderá – podia – por – porque – povo – promeiro – que – quê – qual – qualquer – quando – quem – quieto – são – saber – sem – ser – seu – somente – têm – tal – também – tem – tempo – tenho – tentar – tentaram – tente – tentei – teu – teve – tipo – tive – todos – trabalhar – trabalho – tu – um – uma – umas – uns – usa – usar – valor – veja – ver – verdade – verdadeiro – você –

## A.2 Palavras comum no domínio

função – leia – escreva – programa – dado – verifique

# Apêndice B

# Base de conhecimento do agente inteligente de diálogo

## B.1 Arquivo Alice.xml

A seguir serão descritos os principais atributos do arquivo context.xml.

Código Fonte B.1: Parte da base de conhecimento do Alice.xml

```
1  <category>
2          <pattern>INICIAR</pattern>
3          <template>Olá <get name="nome" />! Vamos resolver o <get name="
               probAtual" />? (SIM/NÃO)</template>
4  </category>
5
6          <category>
7          <pattern>AJUDA</pattern>
8          <template>Você está sentindo dificuldades? Então, <get name="nome
               "/>, posso lhe indicar um problema semelhante? </template>
9  </category>
10
11 <category>
12         <pattern>* SIM *</pattern>
13         <that>VOCE *</that>
14         <template>Por favor, selecione o Problema <get name="
               probMaisSimilar"/>.</template>
15 </category>
16
17 <category>
18         <pattern>* SIM *</pattern>
19         <that>OLA *</that>
20         <template>Ótimo! Nesse problema temos que '<get name="
               probAtualEnunciado"/>'. Você se recorda de ter resolvido algum
                problema semelhante ao <get name="probAtual"/>? (SIM/NÃO) </
               template>
21 </category>
22
23         <category>
24         <pattern>* NAO *</pattern>
25         <that>OLA *</that>
```

```
26          <template>Os problemas resolvidos anteriormente possuem elementos
                que lhe possam auxiliar na resolução do Problema <get name="
                probAtual"/>? (SIM/NÃO) </template>
27  </category>
28
29  <category>
30          <pattern>* SIM *</pattern>
31          <that>OTIMO *</that>
32          <template>Então, por favor, selecione na janela de 'Problemas
                Resolvidos', um problema que vocÃª julga semelhante ao <get
                name="probAtual"/>.</template>
33  </category>
34
35  <category>
36          <pattern>PROBLEMAS SELECIONADOS</pattern>
37          <template>Hum... Você selecionou o problema: <get name="probSel
                "/>. Vamos analisar as semelhanças entre o problema atual e o
                selecionado. Quais características você julga semelhante
                entre o <get name="probAtual"/> e o <get name="probSel"/>? (
                ENUNCIADO, CATEGORIA DO PROBLEMA, PADRÕES DE PROGRAMAÇÃO).</
                template>
38  </category>
39
40          <category>
41          <pattern>* ENUNCIADO *</pattern>
42          <that>HUM *</that>
43          <template>Vejamos os enunciados dos problemas: O <get name="
                probAtual"/> - '<get name="probAtualEnunciado"/>', enquanto o
                <get name="probSel"/> (PROBLEMA SELECIONADO) - '<get name="
                probSelEnunciado"/>'. Pelo enunciado eu considero que estes
                problemas são '<get name="probSelEnunciadoSimilaridade"/>'.</
                template>
44  </category>
45
46          <category>
47          <pattern>* CATEGORIA *</pattern>
48          <that>HUM *</that>
49          <template>O <get name="probAtual"/> Ã© um problema '<get name="
                probAtualCategoria"/>', enquanto o <get name="probSel"/> (
                PROBLEMA SELECIONADO) - '<get name="probSelCategoria"/>'. Pela
                 categoria os problemas são '<get name="
                probSelCategoriaSimilaridade"/>'</template>
50  </category>
51
52  <category>
53          <pattern>* PADROES *</pattern>
54          <that>HUM *</that>
55          <template>O <get name="probAtual"/> é um problema que utiliza os
                padrões '<get name="probAtualPadroes"/>', enquanto o <get name
                ="probSel"/> (PROBLEMA SELECIONADO) - utiliza os padrões '<get
                 name="probSelPadroes"/>'. Pelos padrões de programação os
                problemas são '<get name="probSelPadroesSimilaridade"/>'</
                template>
56  </category>
57
58  <category>
```

```
59          <pattern>* ENUNCIADO *</pattern>
60          <template>Vejamos os enunciados dos problemas: O <get name="
                probAtual"/> - '<get name="probAtualEnunciado"/>', enquanto o
                <get name="probSel"/> (PROBLEMA SELECIONADO) - "<get name="
                probSelEnunciado"/>". Pelo enunciado eu considero que estes
                problemas são '<get name="probSelEnunciadoSimilaridade"/>'.</
                template>
61  </category>
62
63          <category>
64          <pattern>* CATEGORIA *</pattern>
65          <template>O <get name="probAtual"/> é um problema '<get name="
                probAtualCategoria"/>', enquanto o <get name="probSel"/> (
                PROBLEMA SELECIONADO) - '<get name="probSelCategoria"/>'. Pela
                 categoria os problemas são '<get name="
                probSelCategoriaSimilaridade"/>'</template>
66  </category>
67
68  <category>
69          <pattern>* PADROES *</pattern>
70          <template>O <get name="probAtual"/> é um problema que utiliza os
                padrões '<get name="probAtualPadroes"/>', enquanto o <get name
                ="probSel"/> (PROBLEMA SELECIONADO) - utiliza os padrões '<get
                 name="probSelPadroes"/>'. Pelos padrões de programação os
                problemas são '<get name="probSelPadroesSimilaridade"/>'</
                template>
71  </category>
```

## B.2 Arquivo Context.xml

A seguir serão descritos os principais atributos do arquivo context.xml.

Código Fonte B.2: Principais atributos do context.xml

```
1  <!-- Atributos utilizados para troca de mensagens com o RBC -->
2    <set name="nome" value="meu caro aluno"/>
3
4    <set name="probMaisSimilar" value="7"/>
5
6    <set name="probAtual" value="Problema 10"/>
7    <set name="probAtualEnunciado" value="Escreva um programa que some dois
          vetores inteiros do mesmo tamanho gerados aleatoriamente e imprime
         o maior e menor inteiro da soma, indicando se tais números são par
         ou impar. Não use as funções min ou max de python."/>
8    <set name="probAtualCategoria" value="Matemático"/>
9    <set name="probAtualPadroes" value="Repetição Contada, Par, Maior Valor
          e Menor Valor"/>
10
11   <set name="probSel" value="4"/>
12   <set name="probSelEnunciado" value="Escreva uma função que some dois
         vetores inteiros do mesmo tamanho. Com base na função, escreva um
         programa que leia dois vetores da entrada e imprima a soma dos dois
         vetores. O programa deve indicar que os vetores são de dimensões
         diferentes e não executar a soma, se for o caso."/>
```

```
13    <set name="probSelCategoria" value="Matemático"/>
14    <set name="probSelPadroes" value="Seleção simples e Repetição Contada
          "/>
15    <set name="probSelSimilaridade" value="semelhantes"/>
16    <set name="probSelEnunciadoSimilaridade" value="semelhantes"/>
17    <set name="probSelCategoriaSimilaridade" value="idênticos"/>
18    <set name="probSelPadroesSimilaridade" value="semelhantes"/>
```

# Apêndice C

# Questionários e roteiro de avaliação

## C.1 Questionário do Perfil do Aluno

Foi criado um formulário *on-line* no *google Doc* para reconhecer o perfil dos participantes da avaliação. O questionário pode ser acessado em (http://spreadsheets.google.com/viewform?hl=enformkey=cjRtR0hmS1dERll4dWZpRklRMWJVVkE6M
e as respostas visualizadas em (http://spreadsheets.google.com/ccc?key=r4mGHfKWDFYxufiFIQ1bUVAh

# Formulário de Investigação do Perfil do Aluno

* Required

**Nome** *

**Sexo** *

- Masculino
- Feminino

**Idade** *

**Há quanto tempo você programa?** *
Indique quanto tempo de experiência você tem de programação.

- Menos de 1 mês
- 1 a 2 meses
- 2 a 4 meses
- 4 a 6 meses
- 6 meses a 1 ano
- Mais de 1 ano

**Você gosta de programar?** *

- Sim
- Não
- Nem gosto e nem desgosto

**Para você programar é uma atividade.** *

- Muito difícil
- Difícil
- Nem fácil e nem difícil
- Fácil
- Muito fácil

**Qual sua maior dificuldade quando está programando?** *

- ○ Entender o problema
- ○ Criar a solução para o problema
- ○ Utilizar a sintaxe da linguagem na codificação do problema
- ○ Depurar o programa codificado

**Qual estratégia você utiliza para solucionar um problema de programação?** *

- ○ Inicia sua solução a partir do zero
- ○ Recorda de problemas semelhantes e reutiliza esse conhecimento
- ○ Other:

**Você aprendeu a programar por meio de padrões de programação para alunos iniciantes?**

- ○ Sim
- ○ Não

**Selecione o(s) ambiente(s) de programação utilizado(s).**

- ☐ Eclipse IDE
- ☐ ProPAT
- ☐ PyDev
- ☐ Eric Python IDE
- ☐ Wing IDE
- ☐ NetBeans
- ☐ Other:

**Você já utilizou algum ambiente que lhe auxilie na resolução do problemas de programação?** *

- ○ Sim
- ○ Não

Submit

## C.2 Questionário de Avaliação do *Analogus*

Foi criado um formulário *on-line* no *google Doc* para avaliação do *Analogus*. O questionário pode ser acessado em (http://spreadsheets.google.com/viewform?hl=enformkey=cnVhUWpvOUMtekt1bE9xeHFhclpsdGc6MA. e as respostas visualizadas em (http://spreadsheets.google.com/ccc?key=ruaQjo9C-zKulOqxqarZltghl=en).

# Questionário de Avaliação

Esse questionário é composto por um conjunto de questões sobre a ferramenta avaliada. Para cada questão, o usuário deverá informar qual o grau de confiabilidade(0-10) da questão imediatamente anterior.

* Required

**Recordar problemas semelhantes previamente solucionados é uma estratégia interessante para resolver novos problemas.** *

- Concordo plenamente
- Concordo
- Nem concordo nem discordo
- Discordo
- Discordo totalmente

**Confiabilidade** *
Qual o nível de certeza da resposta para questão anterior?

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|
| ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ |

**A ferramenta me ajudou a lembrar de problemas semelhantes ao que está sendo solucionado.**

- Concordo plenamente
- Concordo
- Nem concordo nem discordo
- Discordo
- Discordo totalmente

**Confiabilidade** *
Qual o nível de certeza da resposta para questão anterior?

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|
| ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ |

**A ferramenta indicou, corretamente, problemas com soluções semelhantes ao que estou resolvendo no momento.** *

- Concordo plenamente
- Concordo
- Nem concordo nem discordo
- Discordo
- Discordo totalmente

**Confiabilidade** *
Qual o nível de certeza da resposta para questão anterior?

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|
| ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ |

**Qual o grau de semelhanças entre o problema que foi resolvido e o indicado pela ferramenta?** *

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Diferente | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | Idêntico |

**Você utilizou o conhecimento da solução do problema indicado pela ferramenta para resolver o novo problema?** *

- ○ Sim
- ○ Não

**Os padrões de programação para iniciantes me ajudaram a identificar os problemas semelhantes ao atual.** *

- ○ Concordo plenamente
- ○ Concordo
- ○ Nem concordo nem discordo
- ○ Discordo
- ○ Discordo totalmente

**Confiabilidade** *
Qual o nível de certeza da resposta para questão anterior?

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|
| ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ |

**O professor virtual me ajudou na identificação de problemas semelhantes ao atual.** *

- ○ Concordo plenamente
- ○ Concordo
- ○ Nem concordo nem discordo
- ○ Discordo
- ○ Discordo totalmente

**Confiabilidade** *
Qual o nível de certeza da resposta para questão anterior?

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|
| ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ |

**O professor virtual respondeu de forma satisfatória aos meus questionamentos.** *

- ○ Concordo plenamente

- Concordo
- Nem concordo nem discordo
- Discordo
- Discordo totalmente

**Confiabilidade** *
Qual o nível de certeza da resposta para questão anterior?

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|
| ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ |

**A ferramenta facilitou a utilização dos padrões de programação para alunos iniciantes.** *

- Concordo plenamente
- Concordo
- Nem concordo nem discordo
- Discordo
- Discordo totalmente

**Confiabilidade** *
Qual o nível de certeza da resposta para questão anterior?

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|
| ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ |

**A interface da ferramenta é clara e compreensível.** *

- Concordo plenamente
- Concordo
- Nem concordo nem discordo
- Discordo
- Discordo totalmente

**Confiabilidade** *
Qual o nível de certeza da resposta para questão anterior?

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|
| ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ |

**A ferramenta me ajudou a perceber a importância de solucionar problemas por meio dos problemas previamente resolvidos.** *

- Concordo plenamente
- Concordo
- Nem concordo nem discordo
- Discordo
- Discordo totalmente

**Confiabilidade**
Qual o nível de certeza da resposta para questão anterior?

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|
| ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ |

**A ferramenta me ajudou a refletir sobre os aspectos que tornam os problemas semelhantes.** *

- ○ Concordo plenamente
- ○ Concordo
- ○ Nem concordo nem discordo
- ○ Discordo
- ○ Discordo totalmente

**Confiabilidade**
Qual o nível de certeza da resposta para questão anterior?

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|
| ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ |

**A ferramenta, por meio do professor virtual, estimula a minha habilidade de identificar problemas semelhantes.** *

- ○ Concordo plenamente
- ○ Concordo
- ○ Nem concordo nem discordo
- ○ Discordo
- ○ Discordo totalmente

**Confiabilidade**
Qual o nível de certeza da resposta para questão anterior?

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|
| ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ |

**De um modo geral, a ferramenta me ajuda a solucionar problemas de programação por meio do raciocínio por analogia.** *

- ○ Concordo plenamente
- ○ Concordo
- ○ Nem concordo nem discordo
- ○ Discordo
- ○ Discordo totalmente

**Certeza**
Qual o nível de certeza da resposta para questão anterior?

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ |

**De um modo geral, me sinto satisfeito com a ferramenta.** *

- ○ Concordo plenamente
- ○ Concordo
- ○ Nem concordo nem discordo
- ○ Discordo
- ○ Discordo totalmente

**Confiabilidade** *
Qual o nível de certeza da resposta para questão anterior?

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|
| ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ |

**Recomendaria, sem hesitar, o uso da ferramenta no ensino das disciplinas de introdução à programação.** *

- ○ Concordo plenamente
- ○ Concordo
- ○ Nem concordo nem discordo
- ○ Discordo
- ○ Discordo totalmente

**Confiabilidade** *
Qual o nível de certeza da resposta para questão anterior?

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|
| ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ |

**Sugestões e críticas**

Submit

Powered by Google Docs

Terms of Service - Additional Terms

## C.3 Roteiro de Atividade para Avaliação do *Analogus*

1. Vá em Arquivo>Abrir e selecione o problema "Maior e menor valor do vetor soma".

2. Leia a mensagem do professor virtual na área de diálogo. A mensagem deve ser: "Olá minha caro aluno! Vamos resolver o Problema 10? (SIM/NÃO)".

3. Responda "SIM".

4. O professor virtual informará: "Ótimo! Nesse problema temos que 'Escreva um programa que some dois vetores inteiros do mesmo tamanho gerados aleatoriamente e imprime o maior e menor inteiro da soma, indicando se tais números são par ou impar. Não use as funções min ou max de python.'. Você se recorda de ter resolvido algum problema semelhante ao Problema 10? (SIM/NÃO)".

5. Responda "SIM".

6. O professor virtual informará: "Então, por favor, selecione na janela de 'Problemas Resolvidos', um problema que você julga semelhante ao Problema 10."

7. Vá a área de Problemas Resolvidos e selecione o problema "Frequência das palavras".

8. Visualize a janela flutuante que apareceu. Nela você visualiza as informações do problema selecionado. Na área de codificação, uma nova aba é aberta, informando a solução do problema selecionado.

9. Na área de diálogo, o professor virtual informará: "Hum... Você selecionou o problema: 1. Vamos analisar as semelhanças entre o problema atual e o selecionado. Quais características você julga semelhante entre o Problema 10 e o 9? (ENUNCIADO, CATEGORIA DO PROBLEMA, PADRÕES DE PROGRAMAÇÃO)."

10. Digite "enunciado do problema"e tecle ENTER.

11. O professor virtual informará: "Professor: Vejamos os enunciados dos problemas: O Problema 10 – 'Escreva um programa que some dois vetores inteiros do mesmo tamanho gerados aleatoriamente e imprime o maior e menor inteiro da soma, indicando se tais números são par ou impar. Não use as funções min ou max de python.',

enquanto o 9 (PROBLEMA SELECIONADO) – 'Um professor de português precisa de um programa que dado um parágrafo, apresente a freqüência com que cada palavra se repete neste parágrafo.'. Pelo enunciado eu considero que estes problemas são 'diferentes'."

12. Digite "padrões de programação"e tecle ENTER.

13. O professor virtual informará: "O Problema 10 é um problema 'Repetição Contada, Par, Maior Valor e Menor Valor', enquanto o 9 (PROBLEMA SELECIONADO) - "Repetição com Sentinela". Pelos padrões de programação os problemas são 'diferentes'."

14. Observe o código da solução desse problema na aba "Problema 7"da área de codificação.

15. Vá a área de Problemas Resolvidos e selecione o problema "Somar vetores".

16. Digite "enunciado do problema"e tecle ENTER.

17. O professor virtual informará: "Vejamos os enunciados dos problemas: O Problema 10 - 'Escreva um programa que some dois vetores inteiros do mesmo tamanho gerados aleatoriamente e imprime o maior e menor inteiro da soma, indicando se tais números são par ou impar. Não use as funções min ou max de python.', enquanto o 4 (PROBLEMA SELECIONADO) - 'Escreva uma função que some dois vetores inteiros do mesmo tamanho. Com base na função, escreva um programa que leia dois vetores da entrada e imprima a soma dos dois vetores. O programa deve indicar que os vetores são de dimensões diferentes e não executar a soma, se for o caso.'. Pelo enunciado eu considero que estes problemas são 'semelhantes'."

18. Digite "padrões de programação"e tecle ENTER.

19. O professor virtual informará: "O Problema 10 é um problema 'Repetição Contada, Par, Maior Valor e Menor Valor', enquanto o 4 (PROBLEMA SELECIONADO) - "Seleção simples e Repetição Contada". Pelos padrões de programação os problemas são 'semelhantes'"

20. Digite "categoria"e tecle ENTER.

21. O professor virtual informará: "O Problema 10 é um problema 'Matemático', enquanto o 4 (PROBLEMA SELECIONADO) - 'Matemático'. Pela categoria os problemas são 'idênticos'"

22. Observe o código da solução desse problema na aba "Problema 4"da área de codificação.

23. Vá a área de Problemas Resolvidos e selecione o problema "Par".

24. O professor virtual informará: Você está sentindo dificuldades? Então, meu caro aluno, posso lhe indicar um problema semelhante?

25. Responda "SIM".

26. O professor virtual informará: Professor: Por favor, selecione o Problema 7.

27. O professor virtual informará: "Hum... Você selecionou o problema: 7. Vamos analisar as semelhanças entre o problema atual e o selecionado. Quais características você julga semelhante entre o Problema 10 e o 7? (ENUNCIADO, CATEGORIA DO PROBLEMA, PADRÕES DE PROGRAMAÇÃO)."

28. Digite "enunciado do problema"e tecle ENTER.

29. O professor virtual informará: "Vejamos os enunciados dos problemas: O Problema 10 - 'Escreva um programa que some dois vetores inteiros do mesmo tamanho gerados aleatoriamente e imprime o maior e menor inteiro da soma, indicando se tais números são par ou impar. Não use as funções min ou max de python.', enquanto o 7 (PROBLEMA SELECIONADO) - 'Escreva uma função que receba uma lista de valores numéricos e que retorne, ao mesmo tempo, o menor e o maior número da lista. Não use as funções min ou max de python.'. Pelo enunciado eu considero que estes problemas são 'muito semelhantes'."

30. Digite "categoria"e tecle ENTER.

31. O professor virtual informará: O Problema 10 é um problema 'Matemático', enquanto o 7 (PROBLEMA SELECIONADO) - 'Matemático'. Pela categoria os problemas são 'idêntico'

32. Digite "padrões de programação"e tecle ENTER.

33. O professor virtual informará: "O Problema 10 é um problema 'Repetição Contada, Par, Maior Valor e Menor Valor', enquanto o 7 (PROBLEMA SELECIONADO) - "Maior Valor e Menor Valor". Pelos padrões de programação os problemas são 'semelhante'".

34. Observe o código da solução desse problema na aba "Problema 7"da área de codificação.

35. Codifique a solução do Problema 10.

36. Execute a solução.

37. Salve a solução em Arquivo>Salvar Problema e, em seguida, submeta a solução para avaliação do professor.

## C.4 Publicações

Essa pesquisa possibilitou a publicação em conferências nacionais importantes da área de informática em educação:

- **Raciocínio Baseado em Casos para auxílio a Alunos na Resolução de Problemas por Analogia - Uma abordagem para Representação e Recuperação de Casos** [SJCF08]. Artigo apresentado no XIX Simpósio Brasileiro de Informática na Educação (SBIE), que contém um relato sobre a representação e recuperação de casos no domínio de resolução de problemas de programação.

- ***Analogus*: Um Ambiente para Auxílio ao Ensino de Programação Orientado pelo Raciocínio por Analogia** [**?**] aprovado para o XVII Workshop sobre Educação em Computação (WEI) que apresenta à comunidade um ambiente de resolução de problemas de programação que ajuda o aluno a identificar os problemas resolvidos que

são similares ao atual, por meio de um raciocinador baseado em casos, ao mesmo tempo em que um agente inteligente de diálogo o auxilia a refletir sobre os aspectos de similaridade.