Universidade de Caxias do Sul

Centro de Computação e Tecnologia da Informação - CCTI

Curso de Bacharelado em Ciência da Computação

Tiago Arrosi

Ferramenta de Auxílio à Aprendizagem de Algoritmos – Estudo baseado em Ambientes de Aprendizagem para Áreas Complexas

Caxias do Sul

2009

Tiago Arrosi

Ferramenta de Auxílio à Aprendizagem de Algoritmos – Estudo baseado em Ambientes de Aprendizagem para Áreas Complexas

Trabalho de Conclusão de Curso para obtenção do grau de bacharel em Ciência da Computação da Universidade de Caxias do Sul.

Profa. Carine Geltrudes Webber

Orientadora

Caxias do Sul

2009

AGRADECIMENTOS

A Deus pela vida, pois sem Ele nada seria possível.

Aos meus pais, Joelci e Breatriz, por todo amor e carinho que sempre tiveram comigo. Ambos serão responsáveis por todo o sucesso obtido e cada degrau avançado pro resto da minha vida. Durante todos os anos vocês sempre foram um exemplo de sucesso, conquistas, força e coragem.

Ao meu irmão Wilian, pelo companheirismo e gestos de apoio que me deram força a continuar conquistando objetivos e ultrapassando obstáculos.

A todos os professores da UCS por todo o aprendizado, principalmente a minha orientadora Carine, por toda ajuda durante o desenvolvimento deste trabalho e por sempre ter uma palavra de apoio e incentivo.

E a todos os meus amigos e colegas da UCS que sempre estiverem presente nesta caminhada, dando apoio nos momentos difíceis e nunca negando um pedido de ajuda.

Muito Obrigado a todos vocês!

SUMÁRIO

LISTA DE ABREVIATURAS E SIGLAS

Sigla	Significado em Português
AVA	Ambiente Virtual de Aprendizagem
GOF	Gang of Four
JFLAP	Java Formal Language and Automata Package
MVC	Modal-View-Controller
RIA	Rich Internet Application

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE QUADROS

CÓDIGOS

RESUMO

Primeiramente neste trabalho é apresentado um estudo sobre ambientes virtuais de aprendizagem, mostrando suas características, objetivos e as tecnologias utilizadas para ajudar os alunos no estudo. Após, introduzimos aos ambientes de aprendizagem baseados na resolução de problemas, apresentando algumas classificações, sua arquitetura e também as principais características que tem fundamental importância no ensino aos estudantes.

O presente trabalho de conclusão propõe-se no desenvolvimento de algumas ferramentas que serão integradas a um ambiente de aprendizagem [Zenato, 2009]. Essas ferramentas devem ajudar os alunos que estão começando a programar, no desenvolvimento de algoritmos e no seu entendimento. Para o desenvolvimento destas ferramentas serão utilizadas as linguagens Adobe Flex na interface e Java nas demais camadas da aplicação.

ABSTRACT

First in this paper a study on virtual environments of learning is presented, showing its characteristics, objectives and the used technologies to help the pupils in the study. After we introduced to the learning environments based on problem solving, presenting some classifications, its architecture and key features that have fundamental importance in teaching the students.

The present work of conclusion proposed to develop some tools that will be integrated into a learning environment [Zenato, 2009]. These tools should help students who are starting to program, in the development of algorithms and their understanding. For the development of these tools the languages will be used Adobe Flex in the interface and Java in the too much layers of the application.

1 INTRODUÇÃO

Este trabalho pertence ao domínio da Inteligência Artificial aplicada à Educação, se valendo de tecnologias para o desenvolvimento de interfaces amigáveis, mais especificamente no desenvolvimento de componentes que possam ajudar alunos nos testes e depuração de algoritmos utilizando como linguagem o português estruturado.

Aprender a programar é uma tarefa complicada para a maioria dos estudantes, e a principal causa disso é a falta de habilidade para resolver problemas. A maioria dos estudantes apresenta esta dificuldade em problemas envolvendo cálculos matemáticos e conhecimento de lógica [Gomes et al., 2006].

A aprendizagem através da resolução de problemas é uma prática comum nas áreas ditas complexas, tais como a matemática, a física e a programação de computadores.

A resolução de problemas em áreas complexas pode ser feita passo a passo como em um fluxograma. A solução em forma de fluxograma ajuda os estudantes a compreender e resolver os problemas [Chang et AL, 2006].

Ainda segundo Chang, ambientes de aprendizagem que possuem apenas uma etapa aonde os estudantes devem ler, planejar, desenvolver e testar, não conseguem diagnosticar com precisão a fase na qual o estudante encontra dificuldades. Além

disso, a execução de problemas em apenas uma etapa pode se transformar em uma tarefa desgastante, fazendo com que o aluno se sinta cansado e sem motivação para continuar aprendendo [Chang et al, 2006].

Estudos mostram que uma retroação imediata auxilia o aprendizado [Schulze, 1989]. Para Schulze a cada fase que o aluno executa dentro do sistema ele deveria receber um retorno do sistema. Nesse retorno o sistema deve informar o aluno os erros que ele cometeu e as melhorias que ele poderia fazer na resolução do problema.

Analisando os estudos de [Chang et al, 2006] e [Schulze, 1989] concluímos que um ambiente de aprendizagem baseado na resolução de problemas deve possuir fases curtas na resolução dos problemas e ao final de cada fase fornecer uma retroação ao aluno. A retroação deve ser gerada a partir da analise feita ao final de cada fase concluída pelo aluno.

A maioria dos ambientes de aprendizagem para o ensino da programação foram desenvolvidos para ensinar os estudantes a escrever programas [Kumar, 2005]. Kumar, entretanto se preocupou em desenvolver um ambiente que pretende ajudar os estudantes a compreender o programa que foi escrito e fazer com que os erros sejam encontrados e compreendidos de forma rápida e clara, sem a ajuda do professor.

Segundo Kumar, a maneira mais produtiva de fazer com que o aluno entenda um problema é quando o professor orienta o aluno através de uma estrutura base da solução do problema. A execução de um algoritmo passo a passo é a forma de um ambiente virtual de aprendizagem ajudar o aluno a compreender uma solução

desenvolvida para o problema. Nesse contexto é que o presente trabalho pretende contribuir para a aprendizagem de programação.

1.1 Objetivos do Trabalho

O presente trabalho tem como principal objetivo identificar as melhores formas de fazer com que o aluno possa compreender o algoritmo que desenvolveu em um cenário de resolução de problemas através de um ambiente informatizado de aprendizagem.

Para isto, será desenvolvida uma ferramenta para apoiar alunos durante a depuração do seu código, ajudando a encontrar erros na lógica e facilitando o aprendizado do aluno no desenvolvimento de algoritmos. A ferramenta também deve ajudar os alunos no entendimento do algoritmo escrito.

O ambiente deve possuir uma interface amigável e fácil de utilizar. Serão aplicadas soluções de interface visando facilitar ao máximo o uso do sistema, evitando assim que os alunos percam tempo entendendo uma interface complexa ao invés de focar o seu tempo no aprendizado.

E, por fim, realizar testes com alunos no ambiente de aprendizagem, verificando e avaliando os resultados obtidos.

1.2 Organização do Documento

Este trabalho está organizado em quatro capítulos, como descrito a seguir.

1.2.1 Capítulo 2: Ambientes Informatizados de Aprendizagem

No capítulo 2 é apresentado o conceito de ambientes informatizados de aprendizagem, descrevendo suas características, tecnologias utilizadas no desenvolvimento, arquitetura básica dos ambientes e são testados e analisados ambientes de programação para verificar suas características, principalmente focando a parte de testes e depuração de algoritmos.

1.2.2 Capítulo 3: Projeto da Ferramenta de Depuração de Algoritmos

O capítulo 3 apresenta o projeto da ferramenta de depuração que será desenvolvida. São apresentados os objetivos da ferramenta, opções de depuração que serão desenvolvidas, tecnologias que serão utilizadas no desenvolvimento e quais são os resultados esperados com o desenvolvimento da ferramenta.

1.2.3 Capítulo 4: Implementação

O capítulo 4 apresenta como foi realizada a implementação do depurador de algoritmo. São apresentados todos padrões de projetos utilizados, a modelagem do sistema, os cenários de utilização da ferramenta e a avaliação realizada para validar o depurador de algoritmos.

1.2.4 Capítulo 5: Conclusão

Para finalizar no capítulo 5 são apresentados os aspectos importantes presentes no trabalho, os resultados que foram alcançados e perspectivas de trabalhos futuros e as contribuições do presente trabalho.

2 AMBIENTES INFORMATIZADOS DE APRENDIZAGEM

Ambientes informatizados de aprendizagem são sistemas de computadores que possuem funcionalidades para apoiar processos de ensino e aprendizagem. Eles se utilizam de texto, imagens, animações, sons, vídeos, internet, recursos de interação e de resolução de problemas, para apoiar e motivar o estudante ao longo do processo de aprendizagem [Almeida, 2003] [Brusilovsky, 1992] [Beck et al, 1996].

Inicialmente os ambientes virtuais de aprendizagem foram desenvolvidos com base em quatro estratégias, com relação as suas funcionalidades [Franco et al, 2003]:

- Utilizar elementos existentes na web, como email e grupos de discussões.
- Agregar elementos específicos da informática, como gerenciamento de arquivos e cópias de segurança.
- Desenvolver elementos da atividade educacional, como módulos para conteúdo e avaliação.
- Adicionar elementos de gerenciamento acadêmico sobre cursos, alunos, avaliações e relatórios.

Tais ambientes não constituíam apenas uma cópia de estruturas já existentes, pois eles possuem diversas características próprias que produzem uma grande

diferença na transformação dos processos estabelecidos na Educação [Franco et al, 2003].

Características como a capacidade de apresentar as informações de forma organizada, promover interações entre pessoas, organizar tudo que é produzido de forma a atingir determinados objetivos, são fundamentais hoje em dia nos ambientes de aprendizagem [Almeida, 2003].

Normalmente as atividades de aprendizagem evoluem em ritmos diferentes para cada aluno e o ambiente de trabalho deve se adequar ao nível de evolução em que o aluno se encontrado, para que ele tenha disponível apenas as ferramentas e informações que esteja capacitado a assimilar [Almeida, 2003].

Idealmente os ambientes de aprendizagem devem ser capazes de interpretar cada passo do estudante dentro do sistema durante seu aprendizado e saber em qual nível de aprendizado o usuário se encontra, podendo ajudá-lo da melhor forma quando ele não souber o que fazer.

Um dos principais aspectos de um AVA é o trabalho autônomo, aonde o estudante deve assumir responsabilidade pelo estudo, decidindo quanto tempo vai dedicar ao estudo, em qual intensidade e momento executará as suas tarefas. Mas esta autonomia pode ser limitada quando são definidos prazos para entrega de trabalhos e regras para utilização dos ambientes de aprendizagem [Costa e Franco, 2005].

O surgimento e utilização da internet nos ambientes de aprendizagem trouxeram uma grande avanço aos mesmos e novas funcionalidades foram incorporadas a eles. Uma das principais características é que os estudantes não ficam restritos apenas a consultas de informações, mas eles passem a ser produtores de informações. Diferente do que ocorria quando não existiam ambientes integrados com a internet, onde os alunos apenas entregavam seus trabalhos ao professores, privando assim os outros estudantes de lerem seus trabalhos, neste novo processo todo mundo com acesso a internet pode ler e consultar o material escrito por qualquer estudante [Costa e Franco, 2005].

Apesar de sempre surgirem novas tecnologias, não basta utilizá-las no desenvolvimento do sistema e ignorar as estratégias tradicionais e de sucesso no ensino que vem sendo desenvolvidas e aprimoradas ao longo dos anos. Citamos algumas estratégias como, por exemplo, sistemas de simulações, aprendizado na solução de problemas e manipulação direta, um sistema de aprendizado deve reunir todas essas estratégias [Vanlehn et al, 2005].

2.1 Ambientes de Aprendizagem para Resolução de Problemas

Ambientes de aprendizagem para resolução de problemas são sistemas que fornecem aos estudantes diversos problemas para serem resolvidos. Esses sistemas devem ser capazes de acompanhar e ajudar os estudantes durante a resolução dos problemas. Eles devem fornecer ferramentas, dar dicas para o estudante solucionar os problemas e indicar se a solução do estudante está correta ou não. Alguns sistemas

ainda são capazes de indicar aonde o estudante cometeu o erro quando a sua resposta está incorreta, tais como, o Andes [Gertner & VanLehn, 2000].

O ambiente de aprendizagem Andes é capaz de identificar a parte do problema que o aluno esta com dificuldades e ajudá-lo a resolver apenas esta parte. Com essa ajuda o aluno deve ser capaz de continuar a resolver o problema sozinho [Gertner & VanLehn, 2000].

Ambientes baseados na resolução de problemas podem ser divididos em três grupos baseados na forma com que lidam com os problemas [Kumar, 2005]:

1. No primeiro grupo temos os sistemas que resolvem os problemas informados pelos estudantes. Estes são os sistemas mais flexíveis no sentido de apoio ao aprendizado. Eles devem ser capazes de lidar com problemas formulados incorretamente. Citamos neste grupo o sistema JFLAP [Cavalcante et al., 2004] utilizado no ensino de conceitos sobre autômatos e máquinas de Turing.

2. O segundo grupo é composto pelos sistemas que administram os problemas fornecidos por um instrutor ou professor. A restrição desses sistemas é que eles possuem um repertório limitado de problemas a serem resolvidos. Por exemplo, o software Aplusix [Nicaud, 2002] pertence a essa categoria. O sistema WebToTeach, agora chamado de TuringsCraft [Arnow & Barshay, 1999] também é um exemplo existente nesta categoria.

3. O último grupo é formado pelos ambientes de aprendizagem que são capazes de criar novos problemas para serem resolvidos. Através de *templates* pré-definidos eles são capazes de gerar problemas com respostas diferentes para o mesmo aluno em ocasiões diferentes e também para alunos diferentes na mesma ocasião evitando assim cópias de respostas. Podemos citar neste grupo os sistemas PILOT [Bridgeman et al., 2000] para algoritmos gráficos e Gateway Labs [Baldwin, 1996] para fundamentos matemáticos na Ciência da Computação.

Conforme a separação de ambientes de aprendizagem baseados na resolução de problemas citada, este trabalho se encaixa no segundo grupo, aonde os problemas devem ser cadastrados pelo instrutor. O sistema deve apenas administrar esses problemas e ajudar o estudante a resolvê-los.

Outra forma de classificar os ambientes de aprendizagem baseados na resolução de problemas é através das formas como o sistema consegue validar uma resposta correta para um problema e/ou avaliar a resposta desenvolvida pelo estudante. Podemos citar as diferentes formas [Kumar, 2005]:

- Alguns ambientes comparam a resposta do estudante com a resposta informada pelo professor.

- Alguns sistemas executam a solução do estudante utilizando um sistema de resolução de problemas independente, como por exemplo, um compilador para determinar se a resposta é correta. Neste grupo citamos o sistema TuringsCraft [Arnow & Barshay, 1999].

- Outros sistemas possuem uma ferramenta integrada para solucionar problemas, e utilizam-na para validar a resposta do estudante. Esta ferramenta para solucionar problemas é restrita a uma classe de problemas cadastrados por um professor, mas é capaz de explicar a solução ao estudante. Por exemplo, o sistema JFLAP [Cavalcante et al., 2004].

Para os sistemas baseados na resolução de problemas proverem um real benefício no ensino, devem ser capazes de acompanhar os estudantes durante a resolução de problemas e fornecerem uma retroação que ajude o estudante. Alguns sistemas provêem uma retroação imediata após cada passo que o usuário executa dentro do sistema, outros fornecem a retroação apenas quando o problema é solucionado.

Outro aspecto importante são as formas de fornecer a retroação aos estudantes, os sistemas mais completos são capazes de identificar os pontos aonde o aluno cometeu o erro e explicar o procedimento que deve ser realizado para corrigi-lo.

2.2 Arquiteturas de Ambientes informatizados de Aprendizagem

Ambientes informatizados de aprendizagem foram definidos em estudos com cinco componentes principais: modelo do estudante, módulo pedagógico, domínio de conhecimento, módulo de comunicação e modelo especialista. A Figura 1 mostra a interação entre estes componentes [Beck et al, 1996].

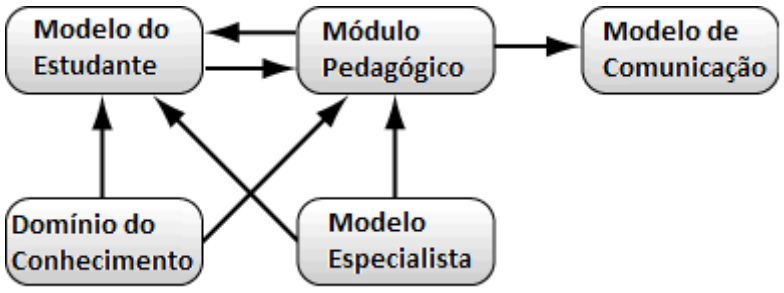


Figura 1 - Interações de componentes em ambientes de aprendizagem

Estes componentes de um ambiente de aprendizagem são descritos a seguir [Beck et al, 1996; Brusilovsky, 1992].

2.2.1 Modelo do Estudante

O modelo do estudante mantém informações sobre cada aprendiz. Informações sobre como o aluno aprende o material que lhe é ensinado até informações sobre equívocos cometidos no seu aprendizado. O propósito deste componente é repassar informações para o módulo pedagógico.

2.2.2 Módulo Pedagógico

O módulo pedagógico provê um modelo do processo de ensino. Ele controla quando o aprendiz deve receber uma revisão da matéria, novos tópicos e qual tópico apresentar.

Este módulo controla o processo em dois níveis diferentes, macro e micro. No nível macro este módulo compila uma lista relevante de operações de ensino e escolhe a melhor operação de ensino no momento. A nível micro o sistema gera um diálogo com a aluno durante a execução de uma operação de ensino selecionada no nível macro.

O modelo do estudante fornece os valores de entrada para este componente, então todas as suas decisões são baseadas nessas informações recebidas.

2.2.3 Domínio do Conhecimento

O componente de domínio do conhecimento possui todo o conhecimento que é passado para os estudantes. Sem este componente o sistema não teria nada para ensinar. Representar este modelo de forma que outras partes do sistema podem acessá-lo é uma tarefa complicada e que exige um grande conhecimento de engenharia.

2.2.4 Módulo de Comunicação

O componente módulo de comunicação é responsável por controlar como as informações são passadas para o estudante. Desde as formas de diálogo com o estudante até os layouts de tela são controlados por este componente.

2.2.5 Modelo Especialista

Assim como o componente de domínio do conhecimento o modelo especialista contém as informações que serão ensinadas aos alunos. Entretanto, nele estas informações são muito mais que apenas representações de dados. Ele é um modelo de como alguém qualificado num domínio específico representa o conhecimento. Utilizando este componente o sistema é capaz de comparar a solução de um estudante com a solução de alguém qualificado, indicando os lugares aonde o estudante possui dificuldades.

2.3 Importância da Interface nos Ambientes Informatizados de Aprendizagem

Os ambientes de aprendizagem devem ser desenvolvidos de forma a facilitar ao máximo o aprendizado do usuário, pois essas ferramentas servem para ajudar os alunos no estudo à distância, inclusive quando o professor não está presente e não há ninguém para auxiliar o aluno.

Segundo Ardito [2004], para oferecer uma experiência recompensadora e ajudar estudantes no seu aprendizado, os ambientes de aprendizagem devem seguir

algumas regras fundamentais: ser interativo, fornecer uma retroação imediata ao usuário, possuir objetivos específicos, motivar, prover uma sensação contínua de desafio e evitar que qualquer fator que possa interromper o aprendizado.

Uma interface limpa, atraente e fácil de utilizar ajuda a manter o aluno sempre motivado. Para Ardito [2004] a interface pode ser listada como um dos fatores podem determinar o sucesso de um sistema de aprendizagem.

Assegurar que os usuários consigam aprender sozinhos em um ambiente de aprendizagem e não encontrem dificuldades em utilizá-lo é uma das tarefas mais difíceis no seu desenvolvimento e poucos conseguem alcançá-la [Ardito et al, 2004].

A interface é uma parte fundamental nos ambientes informatizados de aprendizagem. Contudo mais importante que ser bonito e fácil de utilizar o ambiente deve ser pedagogicamente adequado, segundo Joana Neto “não basta apenas fornecer conteúdos digitais pré-embalados, e não adaptados às nossas especificidades, ou utilizar a Internet para pesquisar/copiar informação sem fazermos qualquer tipo de processamento ou análise sobre o que encontramos, não proporcionará, certamente, aprendizagens significativas” [Neto, 2009].

Ao desenvolver um ambiente de aprendizagem devemos manter a interface mais limpa possível, mas devemos incluir todas as informações que o estudante necessita para resolver o problema apresentado. Tendo toda a informação disponível na interface a carga cognitiva é reduzida, habilitando o estudante a se concentrar apenas na tarefa [Mitrovic et al, 2006].

2.4 Softwares para Aprendizagem de Programação

Existem diversos sistemas que foram desenvolvidos para ajudar os estudantes na aprendizagem da programação de computadores. Cada um foi desenvolvido com um objetivo específico, por isso possuem características que devem ser analisadas. Alguns buscam manter o aluno sempre motivado durante o aprendizado, outros buscam manter a interface simples para que o usuário não perca tempo tentando entender como ela funciona a onde estão as ferramentas que deseja utilizar. Existem também aqueles que mantêm um ambiente focado no trabalho colaborativo para que o aluno sempre aprenda com a ajuda de outras pessoas que estão on-line e estão dispostas a ajudar.

A quantidade de sistemas existentes para ajudar na aprendizagem de algoritmos é vasta, e todos possuem suas qualidades e carências. Analisamos alguns sistemas e realizamos uma comparação entre as funcionalidades que cada um deles possui.

Os sistemas avaliados foram Visualg [Souza, 2009], Eclipse [Eclipse Foundation, 2009], Progranimate [Scott et al., 2008; Progranimate, 2009] e WEBportugol [WEBportugol, 2009]. Eles são descritos a seguir.

2.4.1 Visualg

Visualg é uma ferramenta desenvolvida para interpretar e executar programas escritos em português estruturado, mas com algumas adaptações e novos comandos

criados para facilitar o aprendizado [Souza, 2009]. Veja na Figura 2 a tela principal do Visualg.

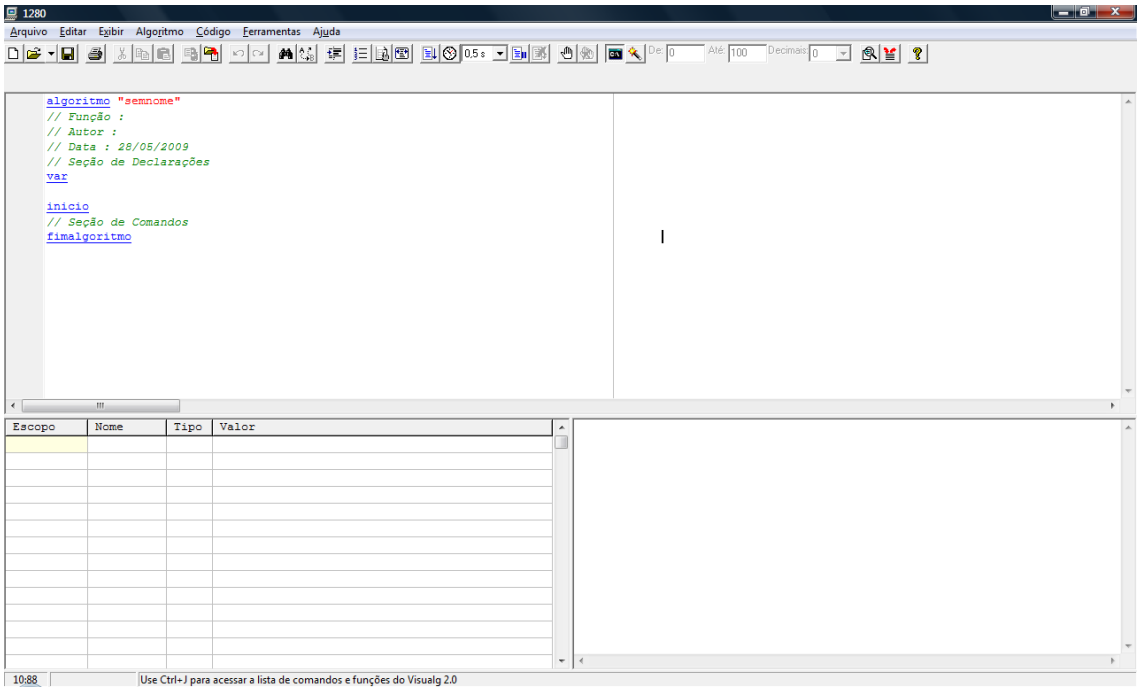


Figura 2 – Tela principal do Visualg

O Visualg possui diversos recursos úteis para quem está aprendendo a programar. Com ele os alunos podem escrever seus algoritmos e executá-los, colocando em prática o que aprendem na sala de aula. A ferramenta também permite aos alunos executar os programas passo a passo, incluir *breakpoints* e também acompanhar o valor das variáveis em cada passo do programa.

Sempre que o aluno solicita a criação de um novo algoritmo a estrutura base já é apresentada no editor facilitando assim o início do algoritmo pelo aluno. Esta estrutura base contém a definição do algoritmo, seção para definições de variáveis e as palavras chaves que indicam o início e fim do algoritmo.

O editor identifica palavras reservadas, grifando elas para que o aluno possa diferenciá-las rapidamente.

Logo abaixo do editor (Figura 2) existem dois quadros. O primeiro apresenta informações das variáveis declaradas no algoritmo e o segundo é um console aonde são apresentados todos os comandos de saída executados durante a execução do algoritmo.

No quadro de variáveis são apresentadas informações como escopo, nome, tipo e valor de cada variável no momento em que se encontra a sua execução, por exemplo, numa execução passo a passo a cada linha executada do algoritmo o quadro de variáveis é atualizado contendo a valor atualizado de cada variável.

Existem diversas forma de execução do algoritmo, a execução pode ser direta aonde não existem paradas durante a sua execução. O aluno pode configurar para o algoritmo ser executado diretamente, mas definindo um tempo para a execução de cada linha do algoritmo. Nessa execução a linha que está sendo executada é realçada em azul para que o aluno identifique o passo que está sendo executado e acompanhe o valor de cada variável no passo atual.

Há também a possibilidade de executar o programa passo a passo aonde o próprio aluno deve comandar o programa informando quando ele deve seguir para o próximo passo.

Em qualquer estilo de execução o aluno pode inserir breakpoints no código dessa forma a execução irá sempre para na linha em que foi colocado o breakpoint e somente seguira em frente quando o aluno solicitar.

Na Figura 3 é possível ver uma imagem de um programa que calcula a fatorial de alguns números sendo executado. Em realce azul a linha que está sendo executada e com realce vermelho um breakpoint inserido no programa. Na parte inferior a esquerda conseguimos acompanhar o valor de cada variável durante a sua execução e a direita um console aonde são apresentadas as mensagens dos comandos de saída.

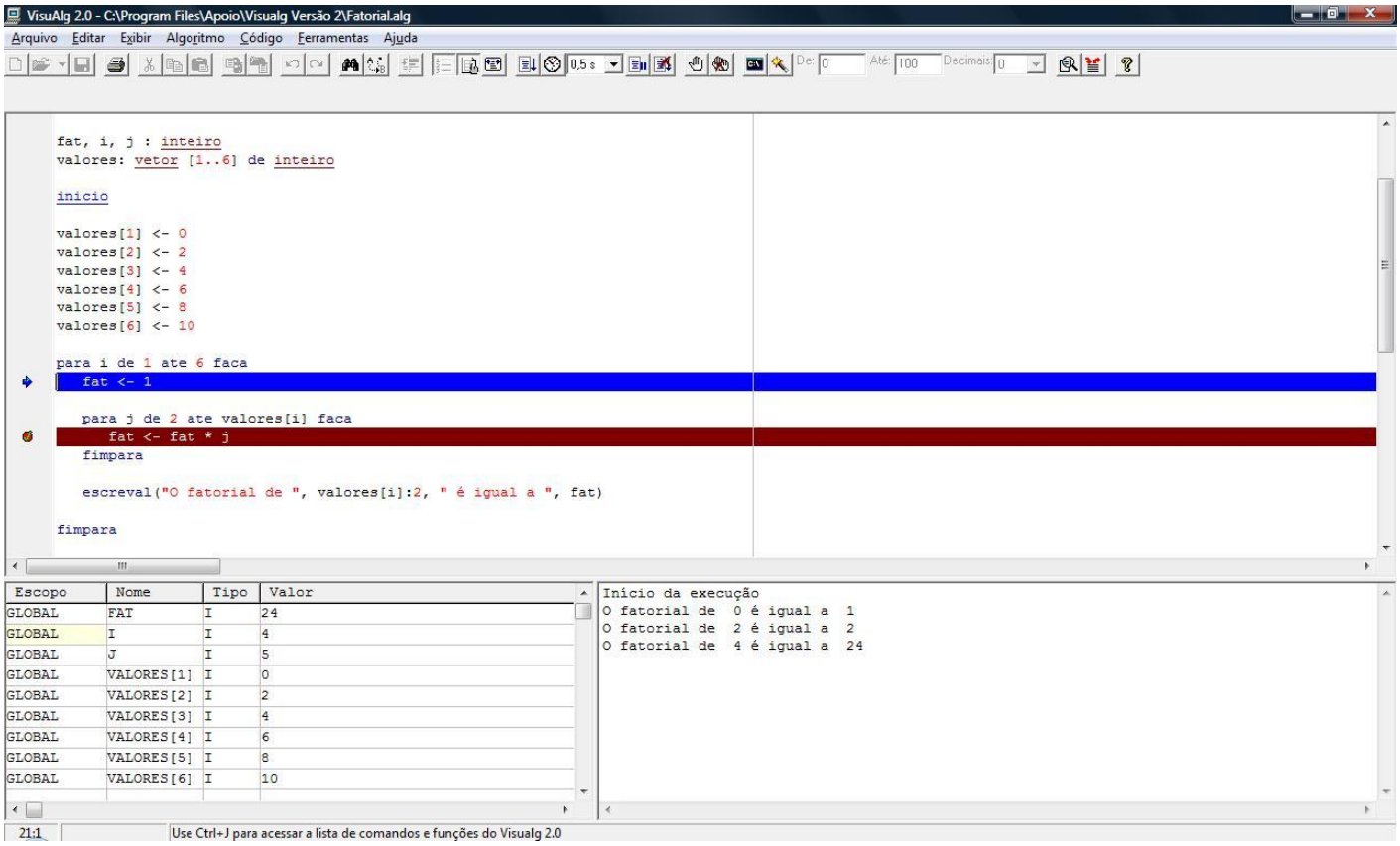


Figura 3 – Algoritmo sendo executado no VisuAlg

O Visualg também é capaz de identificar erros existentes no algoritmo e apresentar de forma clara ao aluno facilitando assim a sua correção. Na Figura 4 verificamos uma mensagem de erro que foi apresentada ao usuário informando que o tipo de dados “inteira” não existe. Na mensagem o usuário pode identificar o número e conteúdo da linha aonde foi encontrado o erro. O conteúdo da linha é editável então o usuário não precisa encerrar a execução do algoritmo para realizar a correção, ele pode corrigir o erro e continuar a sua execução a partir da linha em que o erro foi encontrado. As mensagens de erro são apresentadas durante a execução do algoritmo.

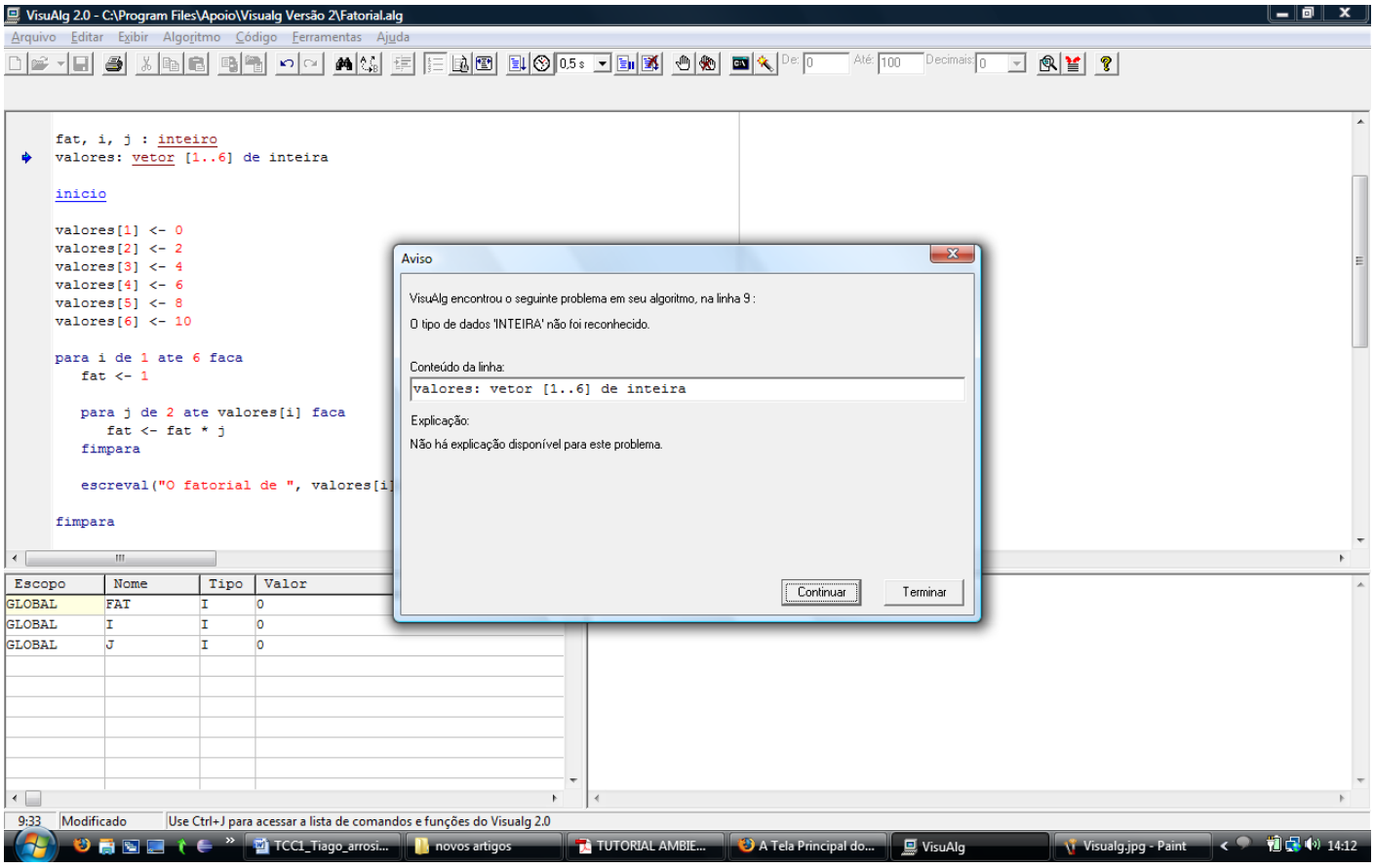


Figura 4 - Mensagem de erro no VisuAlg

2.4.2 Eclipse

O Eclipse [Eclipse Foundation, 2009] não foi desenvolvido pensando na aprendizagem de programação. Esta ferramenta é utilizada no desenvolvimento de sistemas de forma profissional, possuindo assim diversos recursos que auxiliam os programadores no desenvolvimento de sistemas, mas que são muito avançados para os alunos que estão aprendendo a programar.

O ambiente de desenvolvimento do eclipse é totalmente configurável, o usuário pode criar diversas perspectivas de trabalho configurando cada uma da maneira de desejar. É possível incluir as janelas de desejar com as informações que considera importante para o trabalho que está realizando. E estas janelas podem ser inseridas no local de sua preferência, este é um grande diferencial que o eclipse possui em relação às demais ferramentas testadas.

No eclipse existem duas formas de executar um programa, podemos apenas executar o programa aonde serão ignorados os breakpoints inseridos no código, e o programa será executado do início ao fim sem nenhuma interrupção, a não ser que ocorra um erro de execução. Todas as mensagens de saída serão apresentadas ao usuário através da janela chamada de Console.

Pode-se também executar um programa na forma de depuração e configurar o eclipse para trocar a perspectiva, ativando uma previamente configurada contendo apenas as janelas que contém informações importantes para executar uma depuração.

Durante a depuração existem vários ações que podemos realizar, tais como: incluir breakpoints, executar o programa linha por linha, acompanhar as variáveis e seus valores, verificar toda a pilha de execução de métodos, verificar as mensagens de saída, e alterar o valor de qualquer variável em qualquer momento da execução.

Na Figura 5 verificamos a depuração de um programa que realiza o cálculo fatorial de diversos números. Nela verificamos a existência de diversas janelas cada uma contendo uma informação diferente sobre o momento em que a execução do programa se encontra.

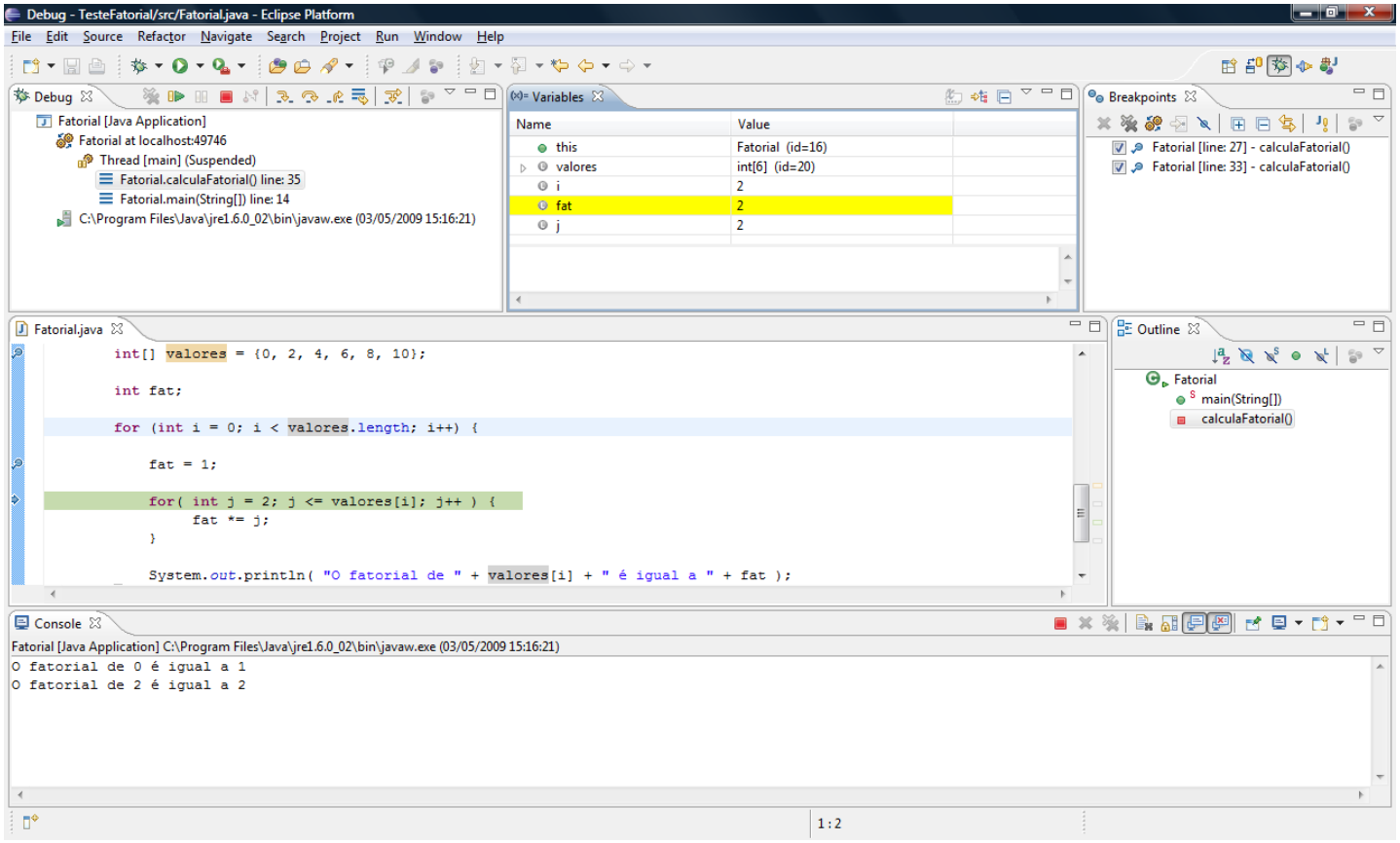


Figura 5 – Depuração de um programa no Eclipse

A janela intitulada “Debug” é a principal em uma depuração, nela existem as ferramentas para a execução do programa e também acompanhar nela uma pilha com todos os métodos que foram chamados.

Na janela de variáveis são apresentadas todas as variáveis do programa contendo o seu valor atual. Em realce amarela identificamos uma variável que foi alterada na última linha executada. Também através dessa janela existe a possibilidade de alterarmos o valor de qualquer variável.

Temos também uma janela chamada breakpoints, aonde são apresentadas todas as condições de parada do programa, através dela executamos diversas ações, tais como: adicionar breakpoints, retirar breakpoints já inseridos, entre outras.

Na parte central da tela temos o código que está sendo executado, nele verificamos que existe uma linha em realce verde indicando a linha que está sendo executada. Na margem da esquerda são identificadas as linhas com breakpoints. Ainda nesta tela podemos verificar o valor de qualquer variável deixando o cursor do mouse sobre a variável que desejar.

Na parte de baixo da tela temos uma janela de console aonde são apresentadas as mensagens de saída incluídas no programa para o usuário.

As mensagens de erro presentes no código são apresentadas no momento que em que usuário vai desenvolvendo o programa, e não durante a sua execução. O eclipse sublinha em vermelho o comando que possui erro e apresentada em uma janela todos os erros encontrados no programa. Na margem esquerda do código existe um ícone identificando o erro e ao clicá-lo são apresentadas algumas soluções para a correção do erro.

Para realizarmos uma demonstração dos erros, inserimos no código uma atribuição para uma variável que não foi declarada no código. Na Figura 6 é possível identificar o erro apresentado na janela de erros e também uma janela com as possíveis soluções para o erro.

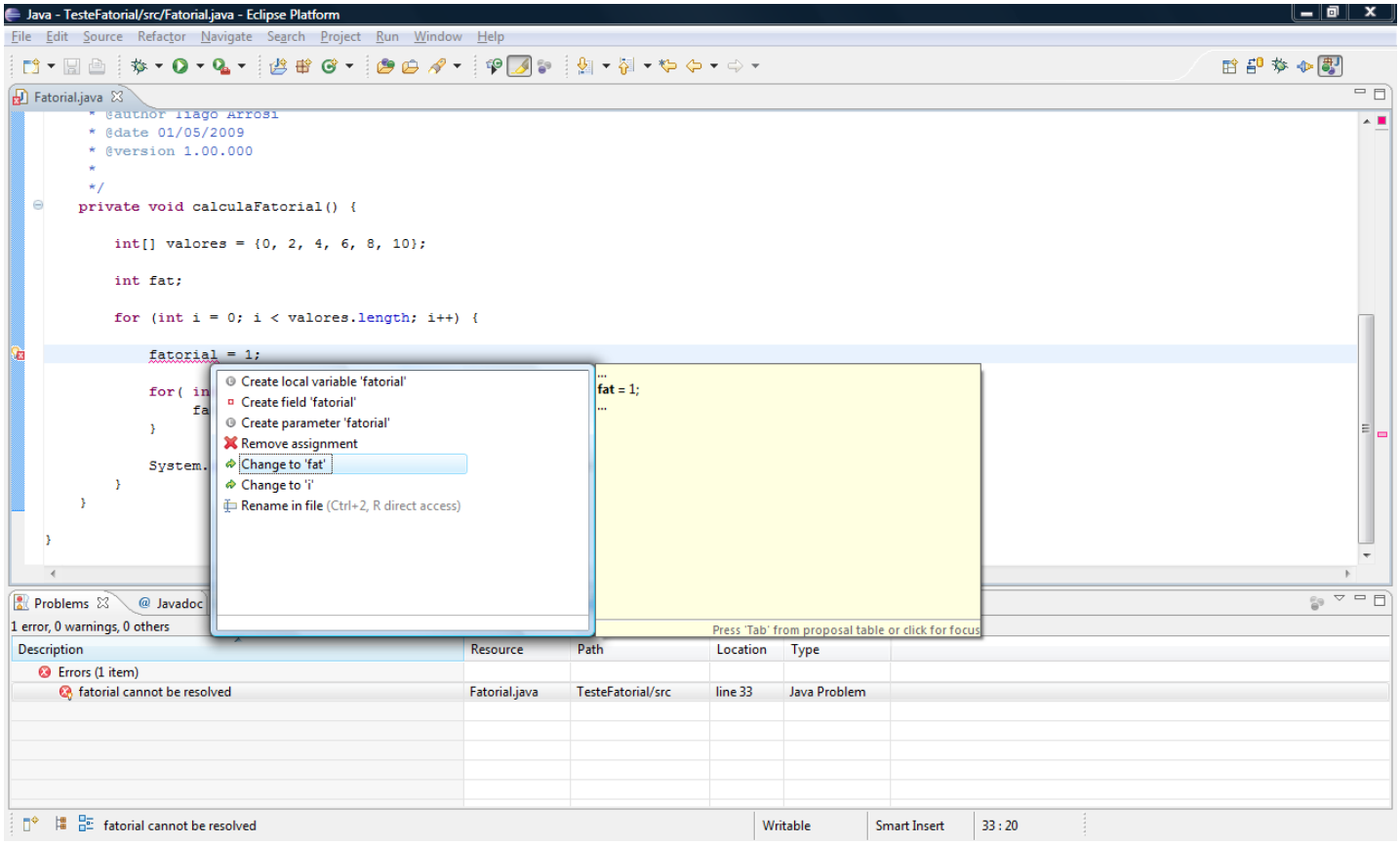


Figura 6 - Mensagem de erro com possíveis soluções no Eclipse

2.4.3 Progranimate

O sistema Progranimate [Scott et AL, 2008] foi desenvolvido para novos programadores aprenderem a programar. Este sistema é diferente dos outros que foram analisados neste trabalho em diversos fatores. Nele o programador não digita o código diretamente em um editor, a programação é feita através de inclusões de componentes em um fluxograma. Sempre que um novo componente é incluído ao fluxograma o código é gerado automaticamente.

A Figura 7 apresenta a inclusão de um comando de repetição no código através do fluxograma. Nele temos que definir a variável de controle, a condição do loop e também o incremento do loop. Podemos verificar também na esquerda todos os componentes que podem ser inseridos no fluxograma para montar o algoritmo.

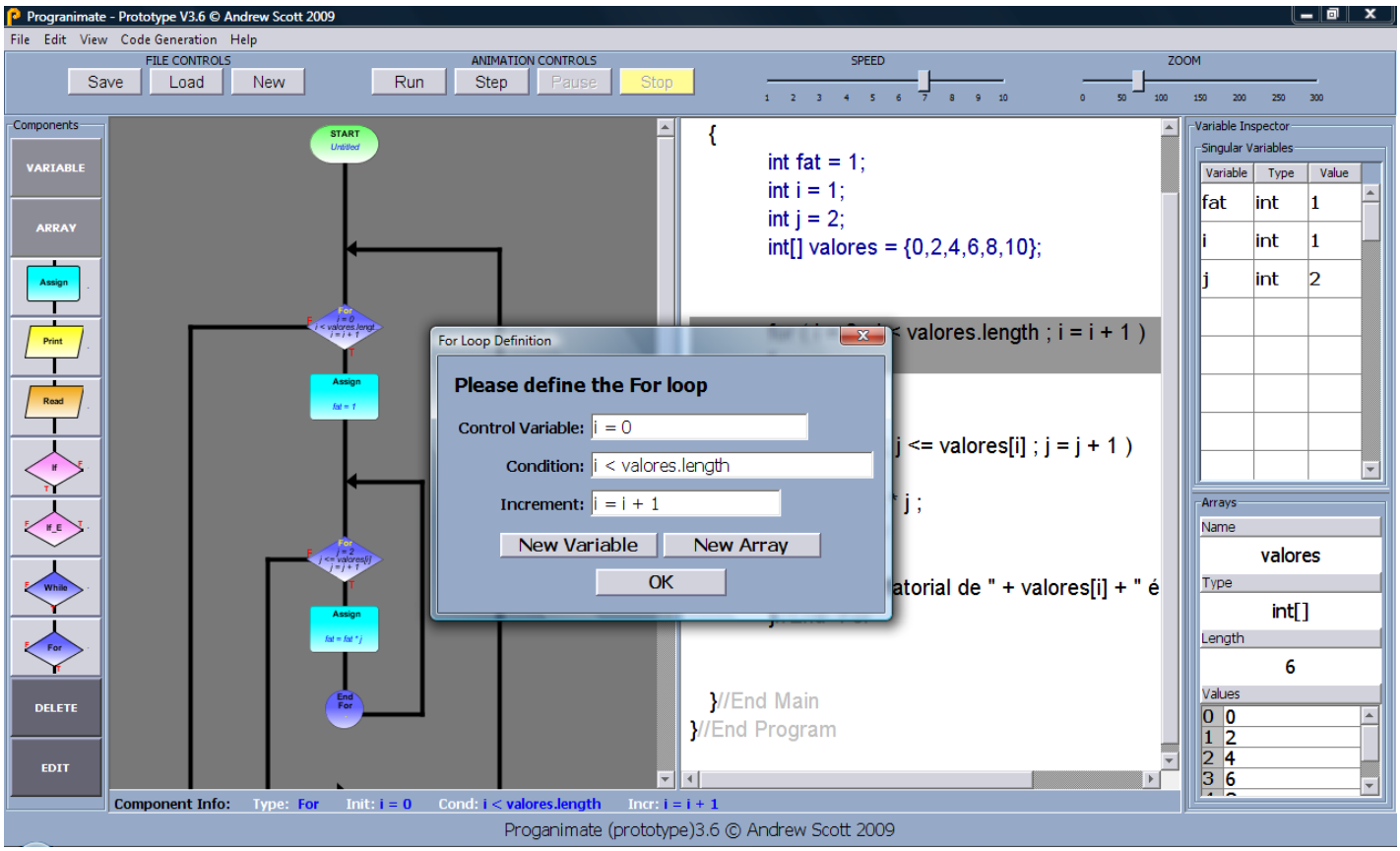


Figura 7 - Inclusão de laço de repetição no Proganimate

Sempre que selecionamos uma linha no código gerado automaticamente o componente relacionado com essa linha é selecionado no fluxograma e vice-versa, mantendo assim uma relação entre os componentes do fluxograma e o código gerado.

Essa sincronização facilita o entendimento do programador em relação ao fluxograma e código gerado.

Neste programa podemos executar os algoritmos passo a passo aonde o usuário manualmente diz quando o programa deve executar a próxima linha ou então executá-lo sem paradas aonde definimos uma velocidade para a execução de cada comando. Neste caso os passos são executados automaticamente pelo sistema.

Verifique na Figura 8 a execução de um algoritmo que realiza o calculo de fatorial para alguns números. A imagem ilustra que está sendo executada uma atribuição e temos o realce do comando no código e no fluxograma para que o programador saiba relacionar o código com o componente no fluxograma.

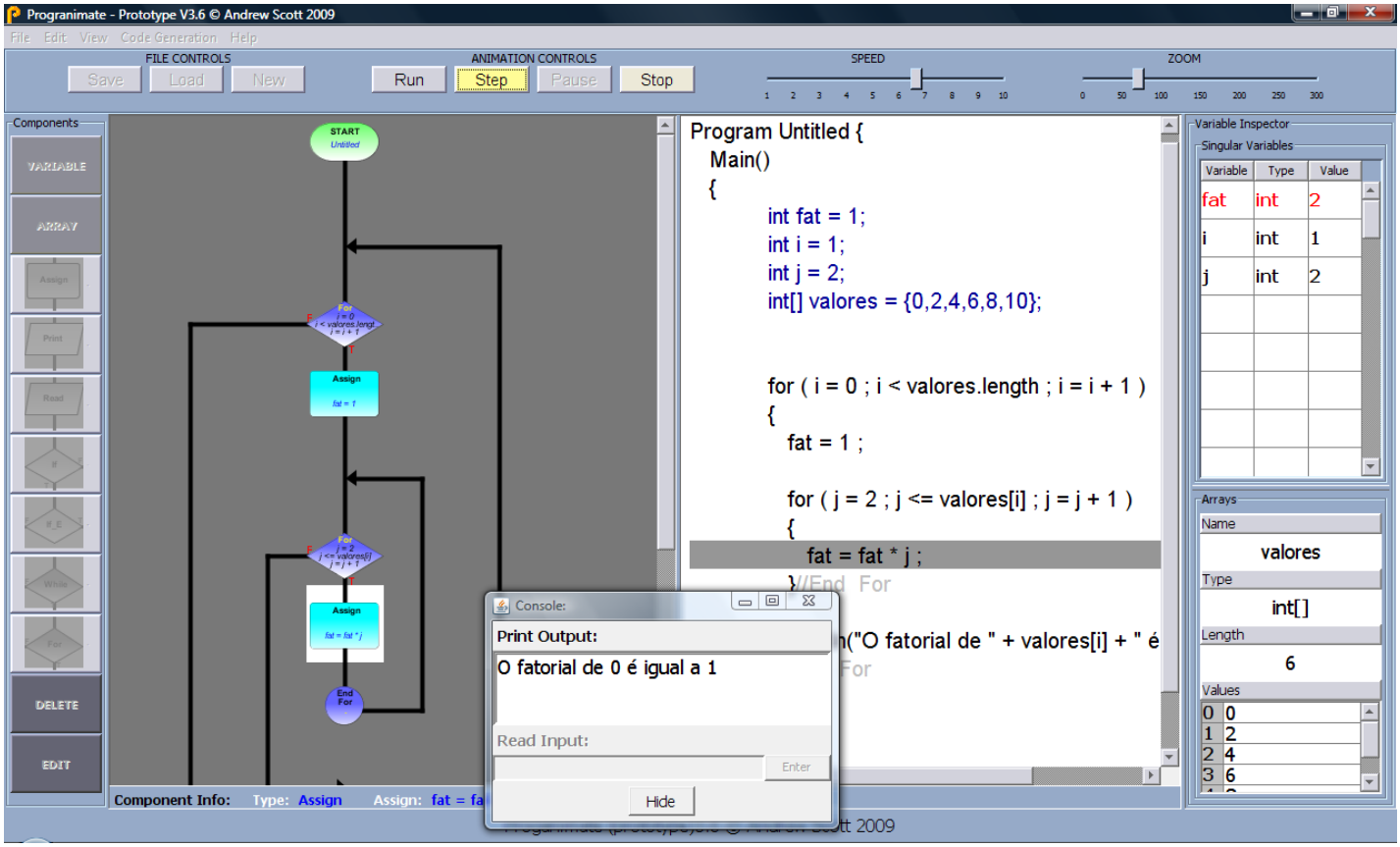


Figura 8 - Execução de algoritmo passo a passo no Programinate

Ao lado direito existe um espaço aonde o usuário pode acompanhar o valor e tipo de cada variável declarada no programa. Sempre que um comando altera o valor de uma variável a mesma fica em vermelho, como na imagem verificamos que a variável “fat” está grifada em vermelho.

Existe também uma janela flutuante aberta sobre as outras onde são apresentadas as mensagens de saída e lidos os valores de entrado do programa.

2.4.4 WEBportugol

Foi analisado também o sistema de ensino a programação chamado WEBportugol [WEBportugol, 2009], dentro todos analisados este é o sistema mais simples e que possui menos ferramentas. Esta ferramenta utiliza a linguagem português estruturado que permite criar programas em português estruturado.

Verificamos na Figura 9 que o sistema possui uma interface simples, limpa e com poucas opções.

Figura 9 - Imagem da interface do WEBportugol

Através do sistema WEBportugol executamos programas de forma corrida sem interrupções e também executá-lo passo a passo, aonde o programador manualmente deve dizer quando o sistema vai executar o próximo comando.

No sistema WEBportual existe também um menu de ajuda aonde o usuário pode acessar o manual completo do sistema e também da linguagem interpretada

pelo mesmo. No menu de ajuda existem também exemplos de alguns programas que podem ser úteis ao programador entender a estrutura de um algoritmo.

Durante a execução do programa são apresentadas novas janelas na sua interface, conforme podemos verificar na Figura 10.

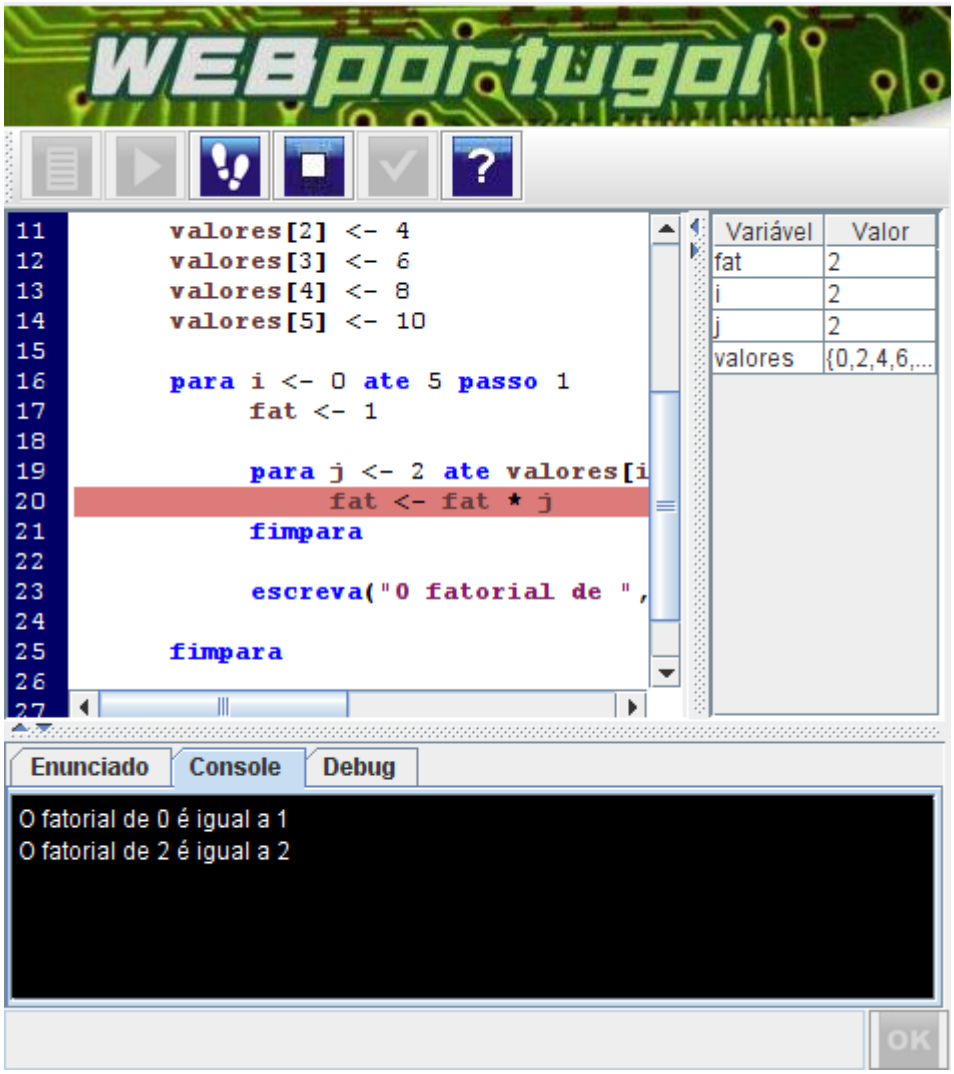


Figura 10 - Execução de algoritmo do WEBportugol

Em uma porção da tela são apresentadas as variáveis definidas no programa e o valor de cada uma no momento em que a execução se encontra. Em outra porção da tela, localizada na parte inferior acompanhamos as mensagens de saída quando são executadas.

Após esta análise constatamos que este é um sistema simples, mas que possui as ferramentas necessárias para o usuário que está aprendendo conseguir desenvolver e testar seus algoritmos.

2.4.5 Quadro Comparativo entre ferramentas analisadas

Para comparar as ferramentas descritas nas seções precedentes elaborou-se a tabela 1. Ela apresenta uma comparação de funcionalidades entre todas as ferramentas que foram testadas e analisadas durante o desenvolvimento deste trabalho.

Analisou-se cada ferramenta com relação aos seguintes critérios:

- **Execução direta**. Capacidade de executar um algoritmo sem paradas e mostrar ao final o resultado da execução e as mensagens inseridas pelo aluno.

- **Execução passo a passo.** Capacidade do ambiente de executar um algoritmo passo a passo, aonde o aluno pode inserir pontos de parada para que a execução seja interrompida para que o aluno continue manualmente.

- **Inspeção de Variáveis.** Área no ambiente aonde o aluno possa acompanhar o valor de cada variável definida durante a execução do algoritmo.

- **Mensagens de erros explicativas.** Mensagens de erro informando o aluno aonde esta o erro e como ele pode resolvê-lo.

- **Indicação de erros durante a programação.** Capacidade do ambiente de verificar durante a implementação do algoritmo os erros cometidos pelo aluno e informar onde eles estão.

- **Solução automática de erros.** Ambiente de aprendizagem fornece ao aluno diversas formas automatizadas para resolver os problemas que foram encontrados.

- **Fluxograma do código.** Ambiente de aprendizagem gera um fluxograma para o algoritmo desenvolvido, ajudando o usuário a entender o seu algoritmo.

- **Ambiente configurável.** Ambiente de aprendizagem pode ter os componentes de interface configurados pelo aluno de acordo com a sua preferência.

- **Realce da Sintaxe.** Editor de algoritmo diferencia palavras reservadas, variáveis e demais comandos de um algoritmo.

- **Alteração nas variáveis durante execução.** Durante a execução de um algoritmo o aluno pode alterar o valor de uma variável.

Tabela 1 – Comparação de funcionalidades entre ferramentas analisadas.

Sistema / Funcionalidades	Visualg	Eclipse	Progranimate	WEBportugol
Execução direta	X	X	X	X
Execução passo a passo	X	X	X	X
Inspeção de Variáveis	X	X	X	X
Mensagens de erro explicativas	X			
Indicação de erros durante programação		X	X	
Solução automática de erros		X		
Fluxograma do código			X	
Ambiente Configurável		X		
Realce da sintaxe	X	X	X	X
Alteração nas variáveis durante execução		X		

2.5 Conclusões Ferramentas de Programação Analisadas

Após a análise de alguns ambientes utilizados na programação foi possível constatar que cada um deles possui características e ferramentas que diferenciam um dos outros.

A ferramenta mais completa sem dúvida é o Eclipse, mas essa é uma ferramenta utilizada para o desenvolvimento profissional. Ela possui inúmeras funcionalidades que estudantes de programação não necessitam, ela é bastante complexa e por isso essa ferramenta não é indicada no ensino da programação.

O Eclipse possui uma característica muito interessante e que não está presente em nenhum outro ambiente de aprendizagem testado, toda a interface é personalizável, com isso o usuário pode customizar a ferramenta deixando ela ajustada da forma que mais lhe agrada.

O ambiente Progranimate possui uma funcionalidade que ajuda muito o aluno no entendimento do algoritmo escrito. O sistema apresenta ao lado do algoritmo um fluxograma que representa visualmente o algoritmo desenvolvido. Este fluxograma ajuda o estudante a visualizar o seu algoritmo e entender como ele é executado. Inclusive durante a depuração do algoritmo passo a passo o sistema marca a linha que está sendo executada no algoritmo e a ação respectiva no fluxograma.

O Visualg é uma ferramenta simples, voltada para a aprendizagem de programação, por isso possui apenas as funcionalidades que o estudante precisa quando esta começando a programar. Nele o estudante consegue: digitar um algoritmo, compilar o algoritmo e verificar se possui erros de sintaxe e executar o algoritmo passo a passo.

O ambiente de aprendizagem WEBportugol é a ferramentas mais simples de todas analisadas, que possui menos funcionalidades e uma interface de tamanho limitado. A única vantagem em relação às demais é que ela foi desenvolvida utilizando applets, podendo então ser executada dentro de um browser e integrada a outros ambientes de aprendizagem que rodam na web.

De todas as ferramentas analisadas a mais indicada para estudantes de programação é a Visualg, pois é uma ferramenta que apresenta apenas as funcionalidades que estudantes de programação irão utilizar. A ferramenta é simples e muito fácil de ser utilizada.

3 PROJETO DA FERRAMENTA PARA DEPURAÇÃO DE ALGORITMOS

Este capítulo apresenta os requisitos do sistema depurador que será desenvolvido neste trabalho. Nele são descritas também as tecnologias que serão utilizadas no desenvolvimento, explicando o motivo da sua utilização. Ao final do capítulo são apresentados os resultados esperados com o projeto e até que ponto de desenvolvimento pretendeu-se atingir.

3.1 Análise de Requisitos

Para o desenvolvimento deste projeto foram levantados alguns requisitos com o principal objetivo de identificar os aspectos mais importantes para ajudar estudantes no entendimento e testes sobre algoritmos desenvolvidos por eles mesmos.

A modelagem de requisitos foi feita através de casos de uso.

3.1.1 Casos de Uso

Os casos de uso modelados são: Depuração de código (Quadro 1), Testes automatizados (Quadro 2) e Fluxograma do Algoritmo (Quadro 3). Eles são descritos nos quadros que seguem.

Quadro 1 – Caso de Uso de Depuração de Código.

Nome do Caso de Uso	Depuração de código
Ator(es)	Estudante
Descrição	Estudante pode executar seu algoritmo passo a passo acompanhando as alterações que sofrem as variáveis definidas durante a execução.
Pré-condições	Algoritmo não pode possuir erros de sintaxe.
Pós-condições	Não há.
Cenário Principal	1. Estudante acessa a depuração de algoritmo. 2. O sistema oferece ao usuário as opções de execução passo a passo e direta sem interrupções. 3. Estudante escolhe o tipo de depuração que deseja executar. 4. Durante a execução do algoritmo, o estudante pode acompanhar os valores de cada variável declarada no algoritmo. 5. Mensagens através de comandos de saída também são apresentadas ao estudante durante a execução.

Cenário Alternativo	**Interromper execução:** 1. Durante execução do algoritmo, o estudante pode interromper a sua execução. **Inclusão de breakpoints:** 1. Antes de iniciar a execução do algoritmo, o estudante pode inserir *breakpoints* (pontos de parada) no algoritmo. 2. Durante a execução direta o sistema verifica se existe *breakpoint* na linha que está sendo executada. Caso encontre, a execução é paralisada e o sitema aguarda por uma ação do usuário para continuar a execução.

Quadro 2 – Caso de Uso de Testes automatizados

Nome do Caso de Uso	Testes automatizados
Ator(es)	Estudante

Descrição	Estudante pode testar o seu algoritmo utilizando dados pré-definidos pelo professor para verificar se a sua solução está correta.
Pré-condições	Algoritmo não pode possuir erros de sintaxe e professor deve definir, juntamente com a definição do problema, valores de entrada e saída do algoritmo que validem a solução do mesmo.
Pós-condições	Não há.
Cenário Principal	1. Estudante executa os testes automatizados. 2. Algoritmo é executado sem paradas utilizando os valores de entrada definidos no problema. 3. Após execução do algoritmo, o sistema compara os valores esperados com os obtidos pelo algoritmo e informa ao estudante se a sua solução está correta ou não.
Cenário Alternativo	Não há.

Quadro 3 – Caso de Uso de Fluxograma do Algoritmo

Nome do Caso de Uso	Fluxograma do Algoritmo

Ator(es)	Estudante
Descrição	Sistema gera um fluxograma a partir do algoritmo e apresenta ao estudante para que ele visualize melhor o seu algoritmo.
Pré-condições	Algoritmo não pode possuir erros de sintaxe.
Pós-condições	Não há.
Cenário Principal	1. Estudante solicita a geração do fluxograma. 2. Sistema gera o fluxograma com base no algoritmo escrito pelo estudante. 3. Fluxograma é apresentado ao estudante.
Cenário Alternativo	Não há.

3.1.2 Diagrama de Caso de Uso

A Figura 10 apresenta o diagrama de caso de uso considerando um aluno em interação com a ferramenta de depuração. A iteração é feita através das três funcionalidades definidas neste trabalho: Depuração de código, Testes Automatizados e Fluxograma do Algoritmo.

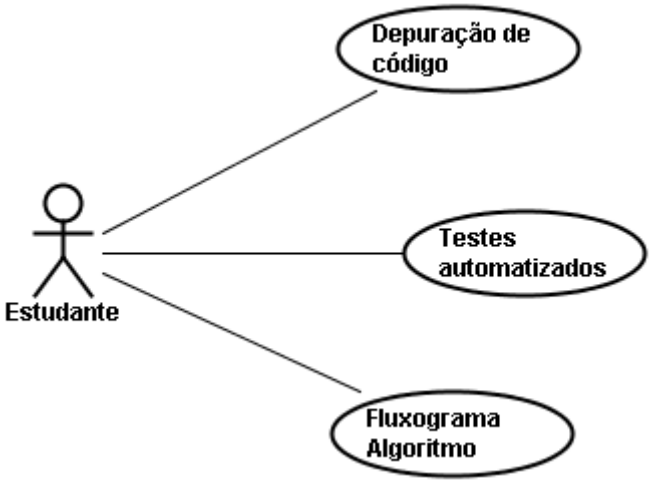


Figura 11 – Diagrama de Caso de Uso

3.2 Projeto

Neste capítulo apresentaremos as informações do projeto em desenvolvimento neste trabalho.

3.2.1 Arquitetura do Projeto

Propõe-se um projeto é dividido em três camadas: camada de apresentação, camada de negócio e camada de persistência. A camada de apresentação é responsável pelos componentes de tela, esta camada não possui nenhuma regra de negócio e não acessa a base diretamente. Esta camada utilizará a tecnologia Flex desenvolvida pela Adobe [Adobe Systems, 2009].

A camada intermediária ou de negócio é aquela onde estão todas as regras de negócios do sistema. Esta camada recebe chamadas apenas da camada de

apresentação e com os dados recebidos realiza as validações e regras de negócio. Esta camada será desenvolvida utilizando a linguagem Java. Não explicaremos com detalhes neste trabalho como será implementada esta camada, pois a sua implementação é feita em projeto de TCC relacionado [Zenato, 2009].

A última camada é responsável pelo acesso a base de dados. Esta camada deve receber requisições apenas da camada de negócio e retornar os dados que forem solicitados, ou realizar a persistência dos dados na base. Nesta camada não deve existir nenhuma regra e nenhuma validação de dados. Esta camada também é implementada utilizando a tecnologia Java. Assim como a camada de negócio a especificação mais detalhada desta camada é explicada em trabalho de conclusão relacionado [Zenato, 2009].

3.2.2 Arquitetura Camada de Apresentação

A camada de apresentação será desenvolvida utilizando o padrão MVC (*Model-View-Controller*). Este padrão teve início com a linguagem Smalltalk, inicialmente para utilizado para mapear a entrada, processamento e saída de tarefas para modelos de interação gráfica com usuário. Apesar de criado há muito tempo, com ele podemos mapear de forma simples no domínio de multicamadas em aplicações empresarias [Sun, 2009].

O padrão MVC possui três grandes componentes: *View*, *Model* e *Controller*. Veja na Figura 12 uma representação da interação entre os componentes desse padrão.

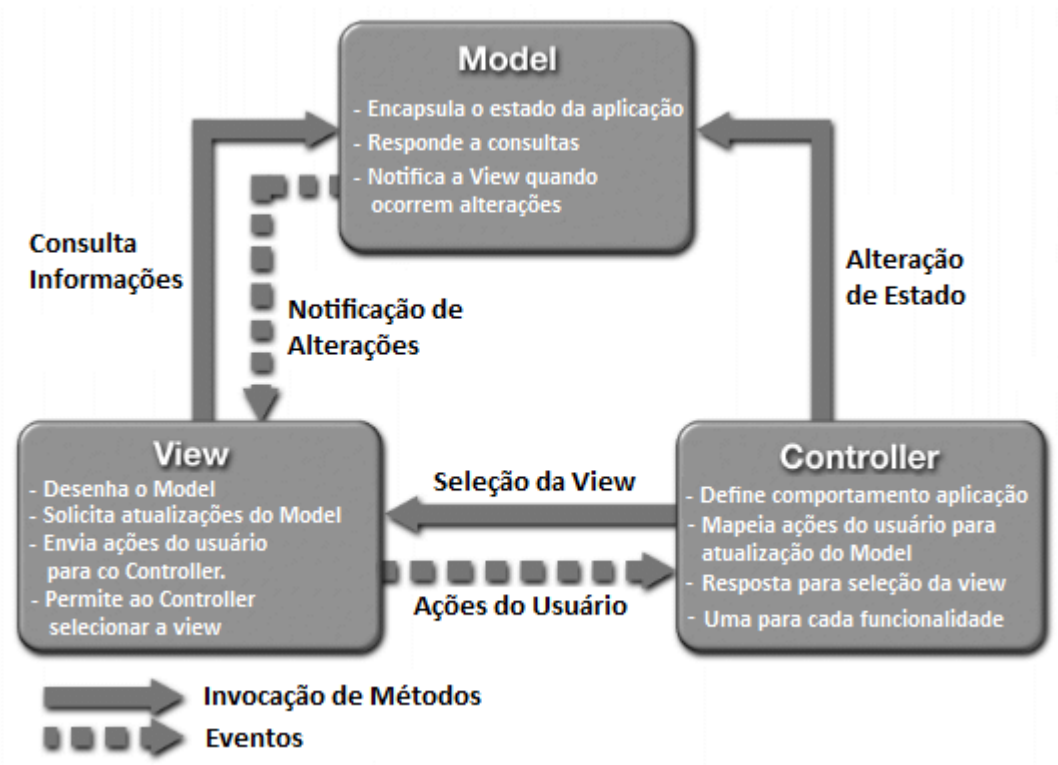


Figura 12 – Padrão Model-View-Controller

O componente *Model* representa o conteúdo que será apresentado ao usuário. Normalmente o *Model* não possui conhecimento da *View*.

O componente *View* desenha os dados do *Model* para melhor visualização do usuário. É de responsabilidade da *View* redesenhar as informações sempre que os dados do *Model* são alterados.

O componente *Controller* responde a ações do usuário sobre a *View*, ele pode inclusive realizar alterações sobre o *Model*. No *Controller* são feitas as validações de dados e alterações de estado da *View* e *Model*.

Este padrão foi escolhido, pois ele possibilita a reutilização de componentes, facilita a inclusão de novos componentes de tela. Devido à separação em componentes, ele facilita também a implementação, testes e manutenção [Sun, 2009].

Além de utilizar o *design pattern* MVC, o projeto fará utilização do padrão Proxy. Os componentes que implementarão este padrão serão responsáveis pelas chamadas ao servidor. O padrão *Proxy* foi definido pelo *Gof* (*Ganf of Four*). Seu objetivo é encapsular um objeto através de outro objeto, sendo o *Proxy*, responsável por controlar o acesso ao primeiro, o objeto real [Gamma et al, 1995].

O *Proxy* esconde o local do objeto real. Para o cliente fica transparente o seu acesso, sem conhecimento se o objeto é remoto ou não [Gamma et al, 1995].

3.2.3 Tecnologia Flex

A camada de apresentação será desenvolvida utilizando a tecnologia Flex [Adobe Systems, 2009]. Flex é uma estrutura de código aberto utilizado, para o desenvolvimento de aplicações Web altamente interativos, desktop e sistema operacional. MXML, uma linguagem declarativa baseada em XML, é utilizada para descrever comportamentos e layout de interfaces. O ActionScript 3, linguagem de programação usada para criar a lógica de cliente [Adobe Systems, 2009].

O Flex possui ainda uma biblioteca com mais de 100 componentes de interface com o usuário, desenvolvidas para a utilização no desenvolvimento de RIA's (*Rich Interface Application*) [Adobe Systems, 2009].

As aplicações Flex são executadas no navegador, utilizando o Adobe Flash Player, ou desktop, através do Adobe AIR. Através do Adobe AIR é possível fazer com que aplicativos Flex acessem dados locais e recursos do sistema no desktop [Adobe Systems, 2009].

A tecnologia Flex foi escolhida para o desenvolvimento deste trabalho por levar a internet todo o poder de interface encontrada em sistemas desktop. Através dela podemos criar componentes mais ricos facilitando a interação do usuário com o sistema.

Outro fator importante que levou a escolha desta tecnologia é a facilidade no desenvolvimento de aplicações Web, principalmente utilizando o Adobe Flex Builder, software que acelera drasticamente o desenvolvimento de aplicações.

O Flex Builder é um ambiente de desenvolvimento baseado na ferramenta Eclipse. Ele agiliza a codificação, possui ferramentas de depuração passo a passo e design visual do *layout* de interface com usuário, aparência e comportamento de RIA. Este ferramenta possui uma licença gratuita para estudantes [Adobe Systems, 2009].

3.3 Compilador

Para alcançar os requisitos levantados temos que implementar um compilador para a linguagem português estruturado, definida para ser a linguagem padrão aceita pelo projeto.

Um compilador é definido como um programa de computador que lê outro programa escrito em certa linguagem e o traduz para outro programa equivalente, porém escrito em outra linguagem, código objeto. Independentemente das linguagens definidas o processo de compilação é o mesmo [Delamaro, 2004*].*

O processo de compilação é composto de análise e síntese. Na análise o objetivo é entender o código fonte e representá-lo em uma estrutura intermediário. A síntese constrói o código objeto a partir da representação intermediário [Delamaro, 2004*].*

A fase de análise é subdividida em análise léxica, sintática e semântica. A síntese é mais flexível e pode ser composta pelas etapas de geração de código intermediário, otimização de código e geração de código final, ou código de máquina. Somente a geração do código de máquina é obrigatória na fase de síntese. A Figura 13 representa o processo de compilação [*Delamaro, 2004*].

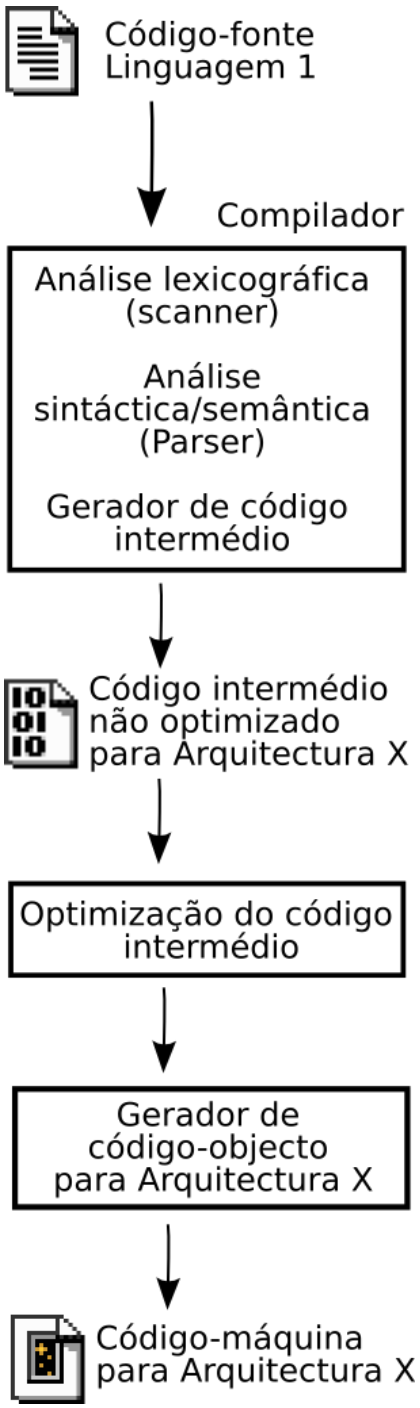


Figura 13 – Processo de Compilação [*Delamaro, 2004*]

3.3.1 Análise Léxica

O analisador léxico recebe de entrada o código fonte de um programa de computador qualquer. O código fonte é lido caractere por caractere e são identificados os elementos básicos de um programa, como: nomes de variáveis, constantes, palavras-chaves, operadores, etc. Durante a análise léxica é eliminado todo o conteúdo irrelevante ao programa, como: espaços em branco, quebras de linhas e comentários. Ao final temos uma seqüência de símbolos chamados de *tokens* [Sguillaro, 2006].

3.3.2 Análise Sintática

A análise sintática recebe na entrada os *tokens* definidos na análise léxica. Esta análise é responsável por verificar a formatação correta dos comandos, de acordo com regras específicas pela gramática da linguagem pré-definida. Quando encontradas sentenças incorretas o processo de compilação é interrompido e são geradas mensagens de erro. Ao final desta fase o programa é representado por uma árvore sintática [Sguillaro, 2006].

A árvore sintática é uma representação aonde os operadores aparecem como nós interiores e os operandos de um operador são os filhos do nó daquele operador. A Figura 14 mostra um exemplo da estrutura de uma árvore sintática.

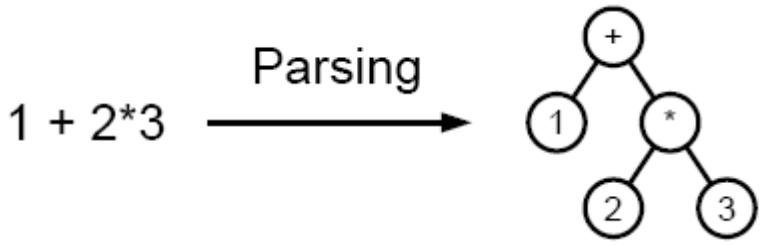


Figura 14 – Representação de uma árvore sintática.

3.3.3 Análise semântica

A análise semântica é a terceira fase do processo de compilação e última fase da análise. Nesta fase são verificados os erros semânticos, por exemplo, verificar se uma multiplicação está sendo feita com dados de mesmo tipo ou de tipos numéricos. Caso encontre alguma inconsistência erros são gerados para avisar o usuário [Sguillaro, 2006].

3.3.4 Geração de código intermediário

Após ser finalizada a fase de análise e não forem encontrados erros, é iniciada a geração de código intermediário. Para realizar a geração do código intermediário, temos que ler a árvore sintática em profundidade (*depth first*), para que o código seja gerado das folhas para os nós. A geração do código é feita utilizando o conceito de quádruplas. Nesse conceito as instruções possuem o seguinte formato: <operador>, <operando 1>, <operando 2>, <resultado> [Sguillaro, 2006].

3.3.5 Implementação do Compilador

Sendo o presente trabalho uma continuação de outro trabalho de conclusão [Possamai, 2008], onde a parte de a compilação já foi implementada. Porém se faz necessária a implementação do código para a geração de código intermediário e execução do algoritmo.

A implementação da geração do código intermediário será realizada na camada Java do sistema. Depois de efetuada a geração do código, as quádruplas geradas serão enviadas para a camada Flex que irá controlar a execução do algoritmo.

Na implementação do código de execução do algoritmo será utilizado o padrão de projeto *Interpreter*, onde cada classe implementa a semântica de um elemento da arvore sintática [Gamma et al, 1995].

3.4 Resultados esperados

Análise e testes sobre ambientes de programação existentes foram feitas, a fim de avaliar seus pontos positivos e negativos. Com os testes realizados também foram levantadas funcionalidades interessantes para quem está iniciando a programar. Foi realizado também um estudo sobre ambientes de aprendizagem baseados na resolução de problemas, este estudo procurou encontrar os problemas e qualidades existentes no ambientes de aprendizagem.

Ao final do processo de análise e pesquisa foram levantadas três idéias de funcionalidades que visam ajudar os estudantes na programação de algoritmos: Depurador de Algoritmo, Execução de testes automatizados e Fluxograma do Algoritmo implementado.

A funcionalidade de depuração de algoritmo foi escolhida, pois é indicado para o estudante executar o seu algoritmo passo a passo e encontrar onde estão os erros existentes. Durante a depuração o estudante pode incluir pontos de parada no seu código, acompanhar a alteração de variáveis e verificar os passos que o seu algoritmo executa para diferentes dados de entrada.

O objetivo da funcionalidade de testes automatizados é que o estudante valide o seu algoritmo de acordo com dados inseridos pelo professor no exercício. Dessa forma, com apenas um clique, o aluno pode rodar o seus algoritmo com dados de entrada pré-definidos e ao final receber uma mensagem informativa dizendo se o algoritmo está correto ou não.

A última funcionalidade escolhida foi o fluxograma do algoritmo desenvolvido. Esta funcionalidade é muito importante para estudantes que não conseguem entender o fluxo do algoritmo que escreveu. Segundo o estudo realizado no presente trabalho, entender o fluxo de um algoritmo é um dos principais problemas encontrados em estudantes que estão iniciando na programação.

Além das funcionalidades levantadas durante a pesquisa foram encontrados alguns problemas existentes nos ambientes de aprendizagem. Esses problemas não

são funcionalidades novas, mas sim comportamentos que estarão presentes no sistema desenvolvido. Os comportamentos que trazem resultados no aprendizado e serão incluídos no sistema são a retroação imediata e também a resolução dos exercícios em diversos passos.

A retroação imediata é importante para que sempre que o aluno cometer um erro ele receba uma mensagem do sistema informando o erro que ele cometeu e como pode resolvê-lo. Dessa forma o estudante não perde muito tempo na mesma situação o que acaba desmotivando o estudante no aprendizado.

A resolução dos exercícios em diversos passos faz com que o aluno possa descansar quando achar necessário, sem perder nada que realizou até o momento. Quando estiver pronto para voltar ao ambiente de aprendizado ele pode continuar do mesmo instante que parou, pois o sistema é capaz de armazenar automaticamente tudo que o estudante já realizou e carregar quando ele volta ao sistema.

Das funcionalidades levantadas foi implementado apenas a depuração de algoritmo. As demais funcionalidades, testes automatizados e fluxograma do algoritmo podem ser implementadas em trabalhos futuros.

4 IMPLEMENTAÇÃO

Este capítulo apresenta a implementação e modelagem da ferramenta de depuração que integra o ambiente de aprendizagem desenvolvido.

4.1 Arquitetura do Depurador de Algoritmos

O projeto do depurador de algoritmos foi desenvolvido em conjunto com um ambiente de aprendizagem [Zenato, 2009]. Para manter uma separação entre os projetos foi definido que o depurador seria desenvolvido dentro do pacote (*package*) "com.ucs.studentbook.debug". Logo, todas as classes deste trabalho estão abaixo desse pacote, exceto as classes que realizam a análise sintática, semântica e léxica, pois elas foram desenvolvidas em projetos anteriores [Rizzoto, 2005; Viganó, 2006; Possamai, 2008].

O depurador desenvolvido possui duas partes principais: A primeira foi desenvolvida utilizando a tecnologia Java. Esta possui a implementação do gerador de código de três endereços. Na segunda parte, desenvolvida utilizando a tecnologia Adobe Flex, foram implementados todos os componentes de interface com o usuário e também a execução do algoritmo implementado.

4.2 Implementação da Geração de Código de Três Endereços

O desenvolvimento da geração do código de três endereços foi desenvolvida sob a tecnologia Java. Esta camada do projeto foi desenvolvida com base em três premissas, que justificam a escolha dos padrões de implementação.

A primeira premissa é que a geração do código de três endereços deveria possuir apenas uma interface de chamada, sendo esta bem definida para quem fosse alterar e utilizar este trabalho. Este ponto de entrada é utilizado para a camada Flex fazer as requisições à camada Java solicitando a geração do código de três endereços. No projeto definimos então que seria implementada uma classe utilizando o padrão de projeto chamado *BusinessDelegate*.

O padrão *BusinessDelegate* é utilizado quando queremos esconder do cliente a complexidade da comunicação remota com os demais componentes [Alur et al, 2004]. A Figura 15 mostra a classe DebugDelegate, implementada neste trabalho seguindo o padrão *BusinessDelegate*. A classe DebugDelegate possui apenas um método chamado debugProgram que recebe uma String com o algoritmo implementado pelo usuário do sistema. Este método não possui lógica nenhuma ele apenas faz a chamada para a classe LexicoSintático que possui as regras de negócio do sistema e faz chamadas para outras classes implementadas no projeto.

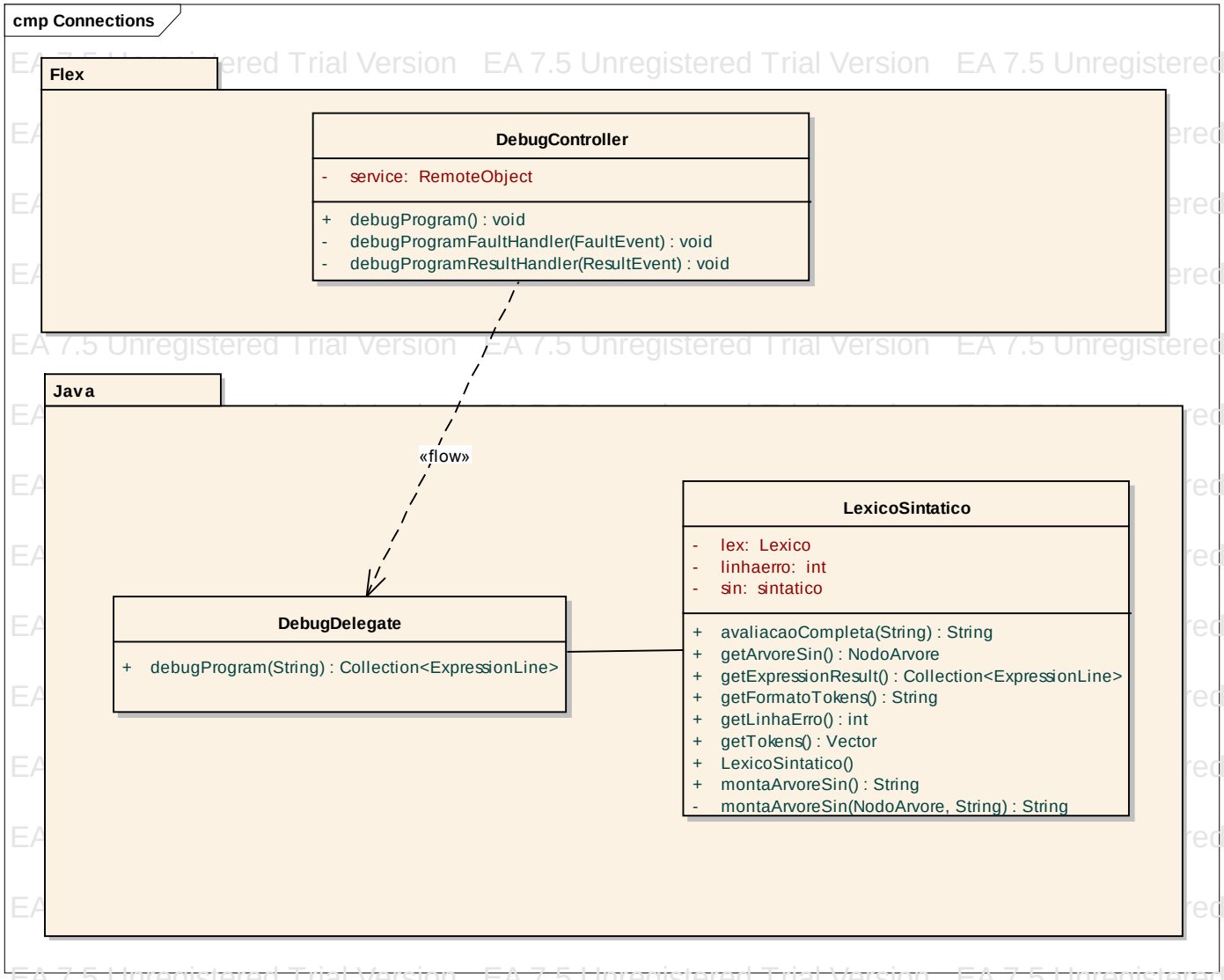


Figura 15 – Classe DebugDelegate ponto de entrada da camada Java no projeto de depuração.

A segunda premissa considerada foi de utilizar a estrutura de dados, montada em trabalhos anteriores, para realizar as análises léxica, sintática e semântica na criação do código de três endereços [Rizzoto, 2006]. O sistema, durante a analise léxica monta uma estrutura de dados chamada *token*, foi utilizada após a análise sintática para a geração do código de três endereços. Dentro da classe denominada sintática, foram criados métodos que são responsáveis por criar a estrutura de dados que será

devolvida a camada de apresentação para ser executado. Veja na Figura 16 como foi defina as classes de análise de algoritmo e geração de código de três endereços para este projeto.

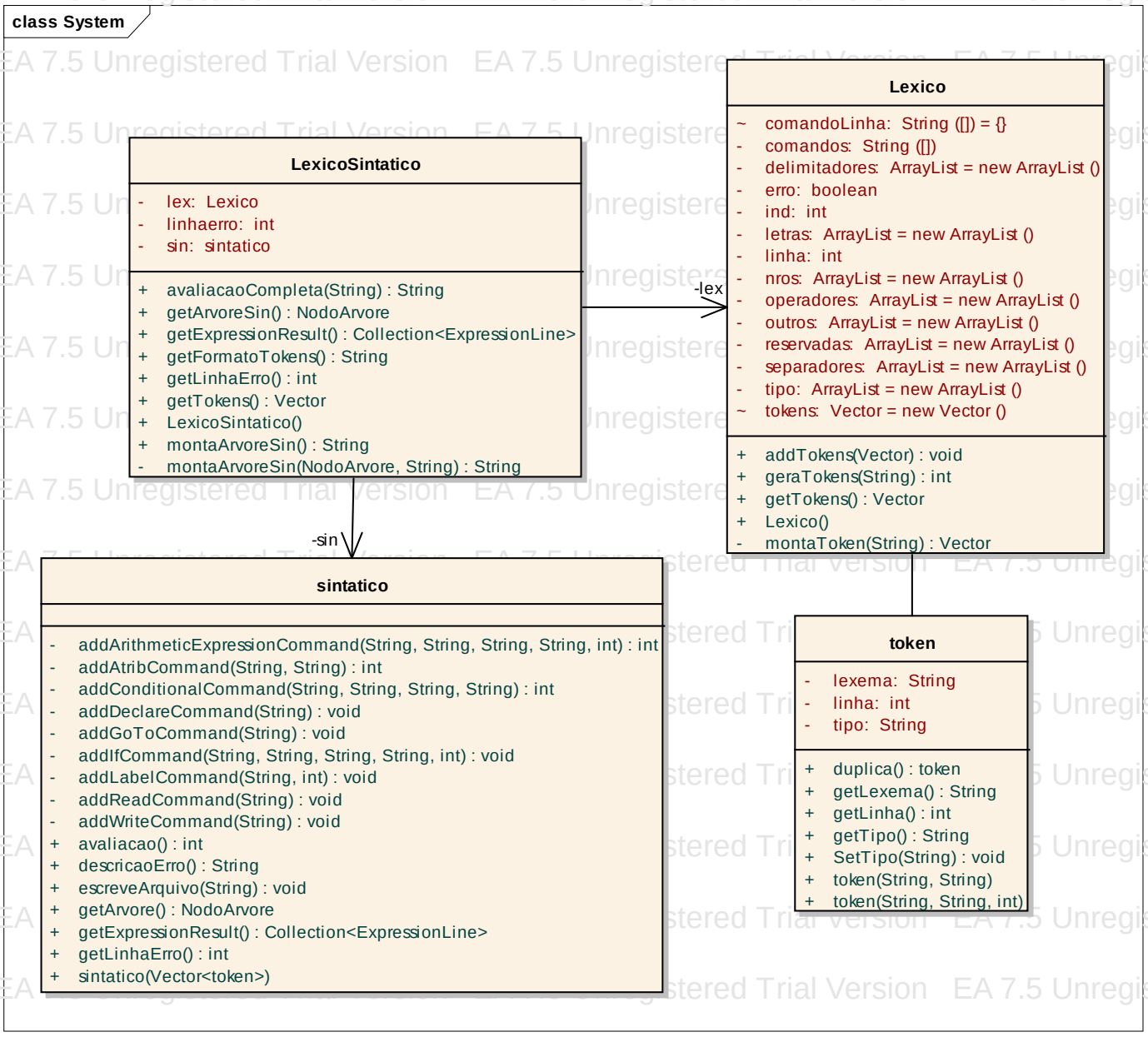


Figura 16 – Classes de análise de algoritmo e geração do código de três endereços.

A terceira e última premissa base para realizar a modelagem dos dados e implementação da camada Java, foi que todas as informações necessárias para executar e depurar o algoritmo deveriam ser estruturadas na camada Java e enviadas para a camada Flex. Essa premissa foi definida para que a camada cliente possua pouco processamento de dados, executando o algoritmo de forma rápida. Para isso o código de três endereços deveria ser mapeado para objetos que possuem todas as informações de cada instrução do algoritmo, inclusive com a linha do algoritmo aonde ele é executado. Pensando nisso criamos uma classe chamada ExpressionLine, esta é a classe que possui toda a informação necessária para a execução de cada comando no Flex. Esta classe possui relacionamento com outras classes que contém outras informações de suporte para a classe ExpressionLine. Veja na Figura 17 a modelagem dessas classes.

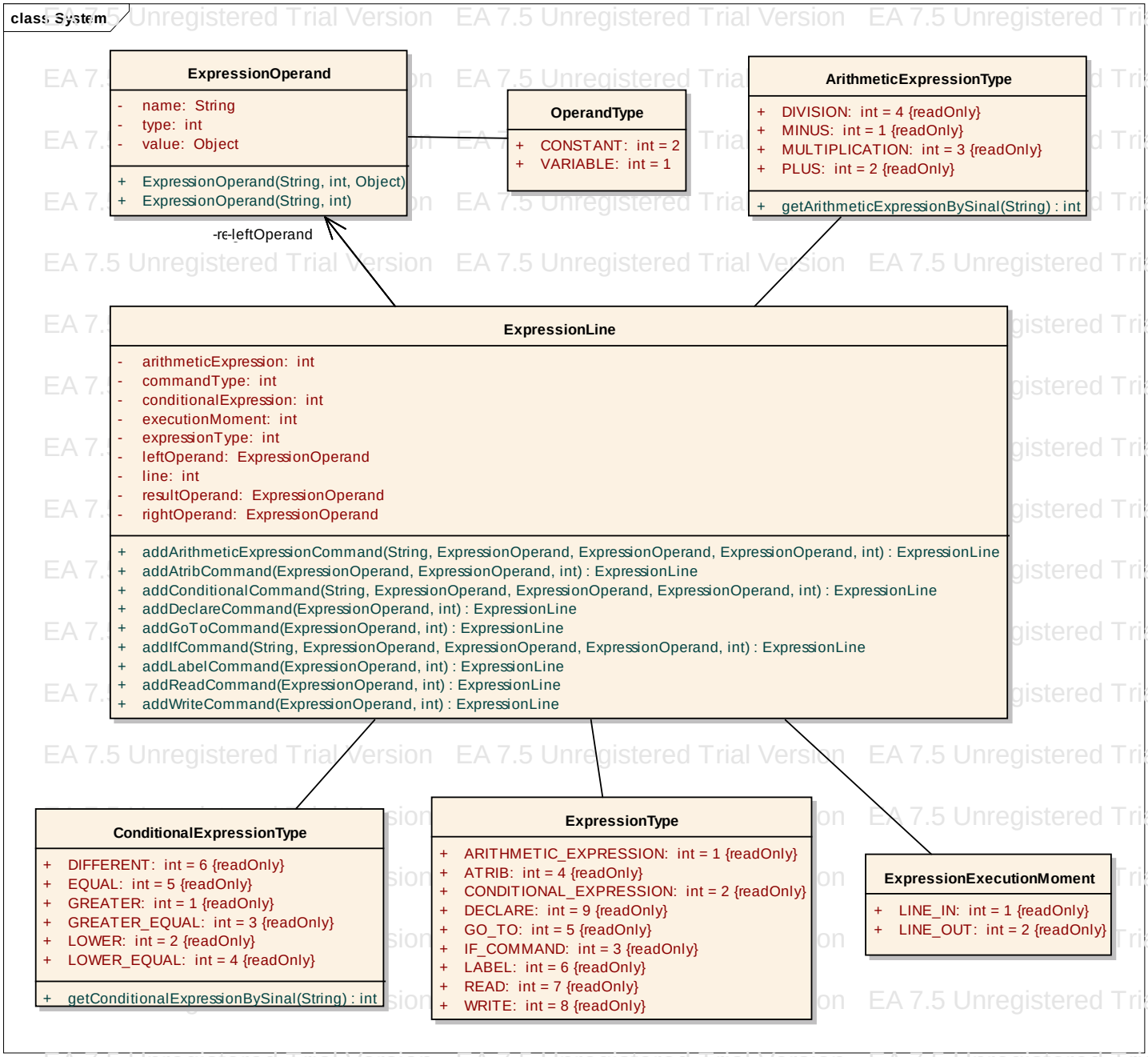


Figura 17 - Classes com informações sobre cada comando que deve ser executado no algoritmo.

4.3 Implementação da Interface de Depuração

A camada de interface com o usuário foi desenvolvida utilizando a tecnologia Flex. Toda interface foi desenvolvida através da criação de componentes reutilizáveis. Para separar o controle, processamento dos dados e apresentação das informações ao cliente foi utilizado o padrão de desenvolvimento MVC (*Model-View-Controller*), apresentado na seção 3.2.2. Para implementar a execução do algoritmo foi utilizado o padrão de projeto chamado *Interpreter* [Gamma et al, 1995].

No presente trabalho modelamos a classe DebugController para ser a responsável pelo controle dos dados e da interface com o usuário. Esta classe possui todos os métodos de execução do algoritmo e controle da interface de acordo com as ações do usuário. Veja na Figura 18 a classe DebugController com seus métodos e atributos.

Figura 18 – Classe DebugController responsável pelo controle dos dados e componentes de interface.

A camada V*iew* do padrão *MVC*, foi implementada de forma a ser composta por diversos componentes de interface. Cada componente sendo responsável por uma tarefa dentro do projeto. Foram definidos os seguintes componentes: DebugLineBase, DebugBarBase, DebugBase, *VariableInspectorBase,ConsoleBase,WriteConsoleMessage, ReadConsoleMessage,* cada um deles responsável por atividade dentro do depurador

de algortitmos. Veja a seguir como cada um deles foi implementado e a sua função principal.

DebugLineBase, componente que representa cada linha do algoritmo no depurador. Através deste componente o usuário é capaz de inserir e remover pontos de parada, verificar a linha que está sendo executada e verificar o código de cada linha do algoritmo.

O componente *DebugBarBase* é o responsável por possuir todas as ações que o usuário pode tomar sobre a depuração do algoritmo, como: executar, parar, interromper a execução do algoritmo, inclusive definindo a velocidade com que o algoritmo é executado.

DebugBase, é o componente que agrupa todos os componentes *DebugLineBase* criados para cada linha do algoritmo e *DebugBarBase* para o usuário controlar a execução do algoritmo.

Veja na Figura 19 abaixo a associção entre esses componentes.

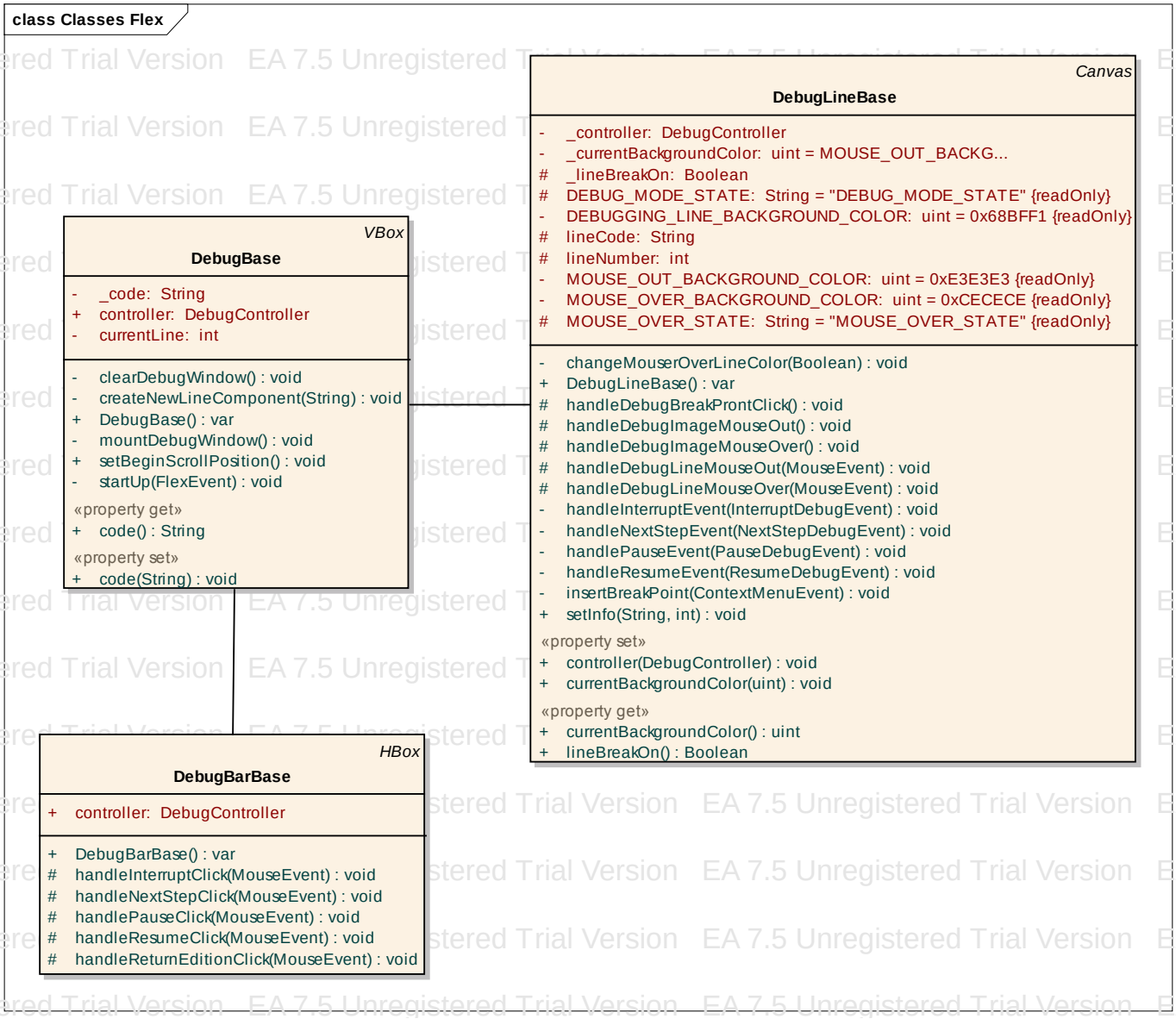


Figura 19 – Associação entre os componentes DebugBase, DebugBarBase e DebugLineBase.

Para apresentar as variáveis do algoritmo ao usuário, com seu valores e ainda permitir que o usuário altere os valores durante a execução do algoritmo foi criado um componente chamado *VariableInspectorBase*, este componente é composto por outro componente chamado *CustomRowColorDataGrid* que é uma tabela aonde é possível colorir as linhas com cores. Este componente foi utilizado para que seja apresentada

ao usuário uma linha com cor diferente para mostrá-lo que a variável daquela linha teve seu valor alterado. Veja na Figura 20 a modelagem do componente de inspeção de vaiáveis.

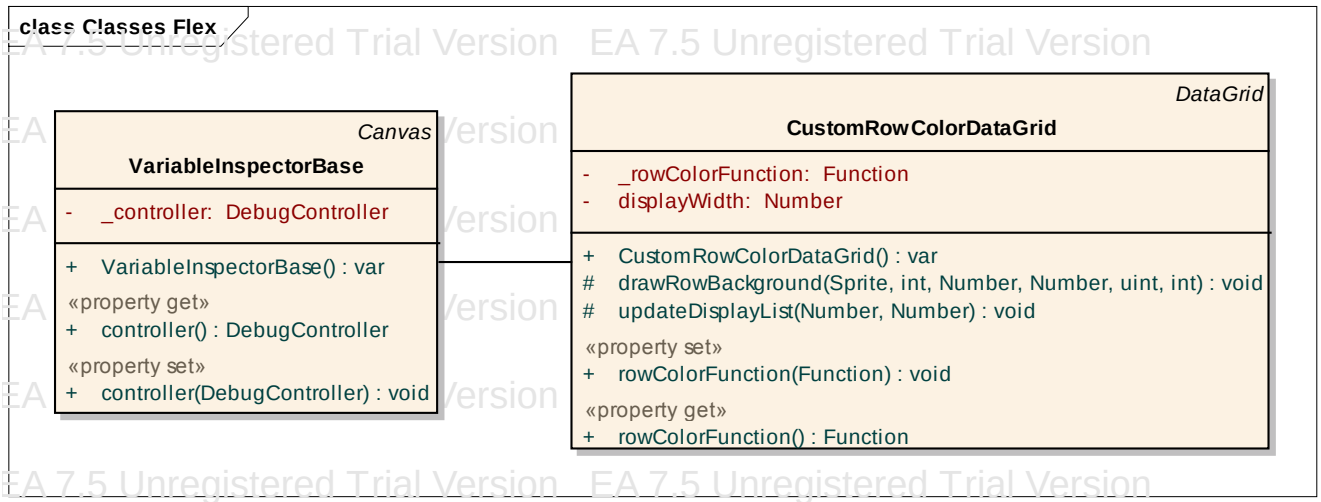


Figura 20 – Modelagem do componente de inspeção de variáveis.

Por último temos o componente aonde são apresentadas as mensagens ao usuário. Este componente foi separado em outros três. O primeiro é o componente *WriteConsoleMessage*, com esse componente apresentamos mensagens ao usuário quando um comando de escreva é executado no algoritmo escrito. O segundo componente é o *ReadConsoleMessage*, este componente é utilizado quando precisamos receber alguma informação do usuário para prosseguir a execução do algoritmo. E temos um último componente chamado de *ConsoleBase* que é o componente que agrupa todas as mensagens de escrita e leitura que serão apresentadas ao usuário. Veja na Figura 21 a modelagem destes componentes.

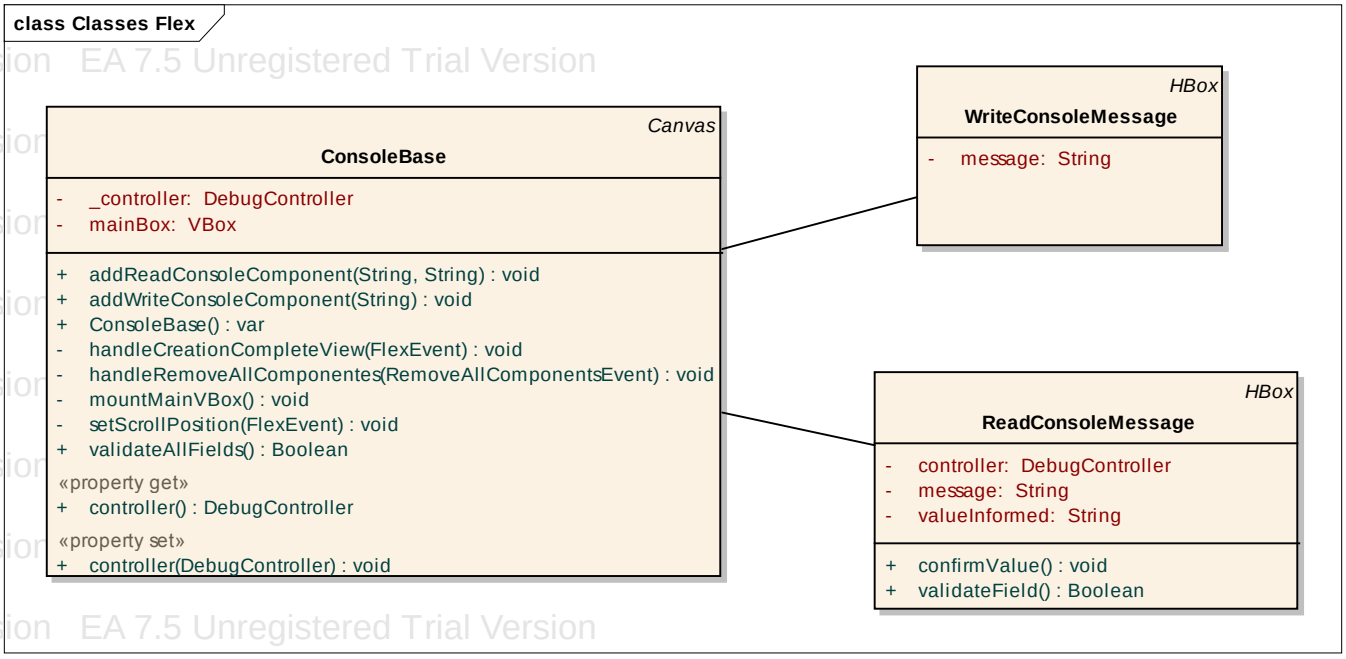


Figura 21 – Modelagem dos componentes de leitura e escrita de mensagens.

Para desenvolver as classes responsáveis pela execução do algoritmo utilizamos o padrão de projeto definido como *Interpreter*. A estrutura desse padrão de projeto pode ser visualizada através da Figura 22 [Gamma et al, 1995].

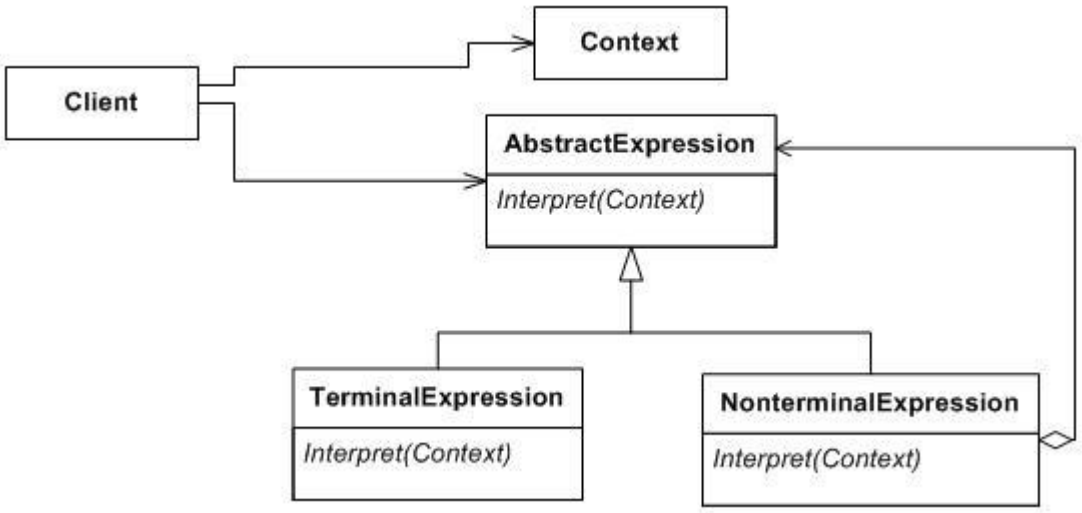


Figura 22 – Estrutura do Padrão de Projeto Interpreter

No padrão *Interpreter* temos uma classe abstrata chamada de *AbstractExpression*, nela é definido uma método abstrato chamado *interpret* que será implementado por todas as classes que estendem esta classe. Na linguagem *ActionScript* não temos como definir classes abstratas então essa classe foi definida como uma interface. No projeto a classe foi chamada de *Expression* e seu código pode ser visto Código 1.

Código 1 – Interface Expression

```
package com.ucs.studentbook.debug.interpreter.expression
{
    import com.ucs.studentbook.debug.interpreter.context.DebugContext;

    public interface Expression {

        function interpret(context:DebugContext) : Object;

    }
}
```

O padrão *Interpreter* também definiu que devemos criar classes, que estendam a classe *AbstractExpression* e implementam o método *interpret*, para todas expressões terminal e não-terminal existente na gramática [Gamma et al, 1995].

Expressões terminais é o menor nível de expressões existentes na gramática, uma expressão é considerada terminal quando não podem mais ser divididas em partes menores, neste projeto as expressões terminais são: variáveis, números, literais e valores booleanos. Veja no Código 2 a implementação da classe *Variable* que defini uma variável no presente trabalho.

Código 2 – Classe que defini uma variável no padrão Interpreter.

```
package com.ucs.studentbook.debug.interpreter
{
    import com.ucs.studentbook.debug.interpreter.context.DebugContext;
    import com.ucs.studentbook.debug.interpreter.expression.Expression;

    public class Variable implements Expression {

        private var _name:String;
        private var _showInspector:Boolean;

        public function Variable(name:String, showInspector:Boolean) {
            _name = name
            _showInspector = showInspector;
        }

        public function interpret(context:DebugContext) : Object {
            return context.getValue(name);
        }

        public function get name() : String {
            return _name;
        }

        public function get showInspector() : Boolean {
            return _showInspector;
        }

    }
}
```

As expressões não terminais são regras de produção dentro da gramática, consistem em uma seqüência de símbolos terminais e não terminais. Neste trabalho definimos diversas expressões não terminais, tais como: expressões numéricas, expressões lógicas, comandos de escrita e leitura, entre outros. Veja no Código 3 a implementação da classe *PlusExpression*, que realiza a soma entre duas variáveis.

Código 3 – Classe que implementa a expressão de soma definida na gramática.

```
package com.ucs.studentbook.debug.interpreter.expression.arithmetic
{
    import com.ucs.studentbook.debug.interpreter.Variable;
    import com.ucs.studentbook.debug.interpreter.VariableType;
    import com.ucs.studentbook.debug.interpreter.context.DebugContext;
    import com.ucs.studentbook.debug.interpreter.expression.Expression;

    public class PlusExpression implements Expression {

        private var _variable1:Variable;
        private var _variable2:Variable;

        public function PlusExpression(variable1:Variable, variable2:Variable) {
            _variable1 = variable1;
            _variable2 = variable2;
        }

        public function interpret(context:DebugContext):Object {
            var first:String = String(_variable1.interpret(context));
            var second:String = String(_variable2.interpret(context));

            if (VariableType.isNumeric(first) && VariableType.isNumeric(second)) {
                return Number(first) + Number(second);
            } else {
                return (first + second);
            }
        }
    }
}
```

Ainda seguindo a implementação do padrão *Interpreter*, temos a classe *Context*, esta classe possui informações globais utilizadas pelo interpretador [Gamma et al, 1995]. No projeto do presente trabalho temos duas classes que fazem parte dessa implementação, são elas a *ContextVariable*, quem contém as informações de todas as variáveis no contexto de execução e a classe *DebugContext* que possui informações gerais da execução do algoritmo. Veja na Figura 23 a modelagem destas classes.

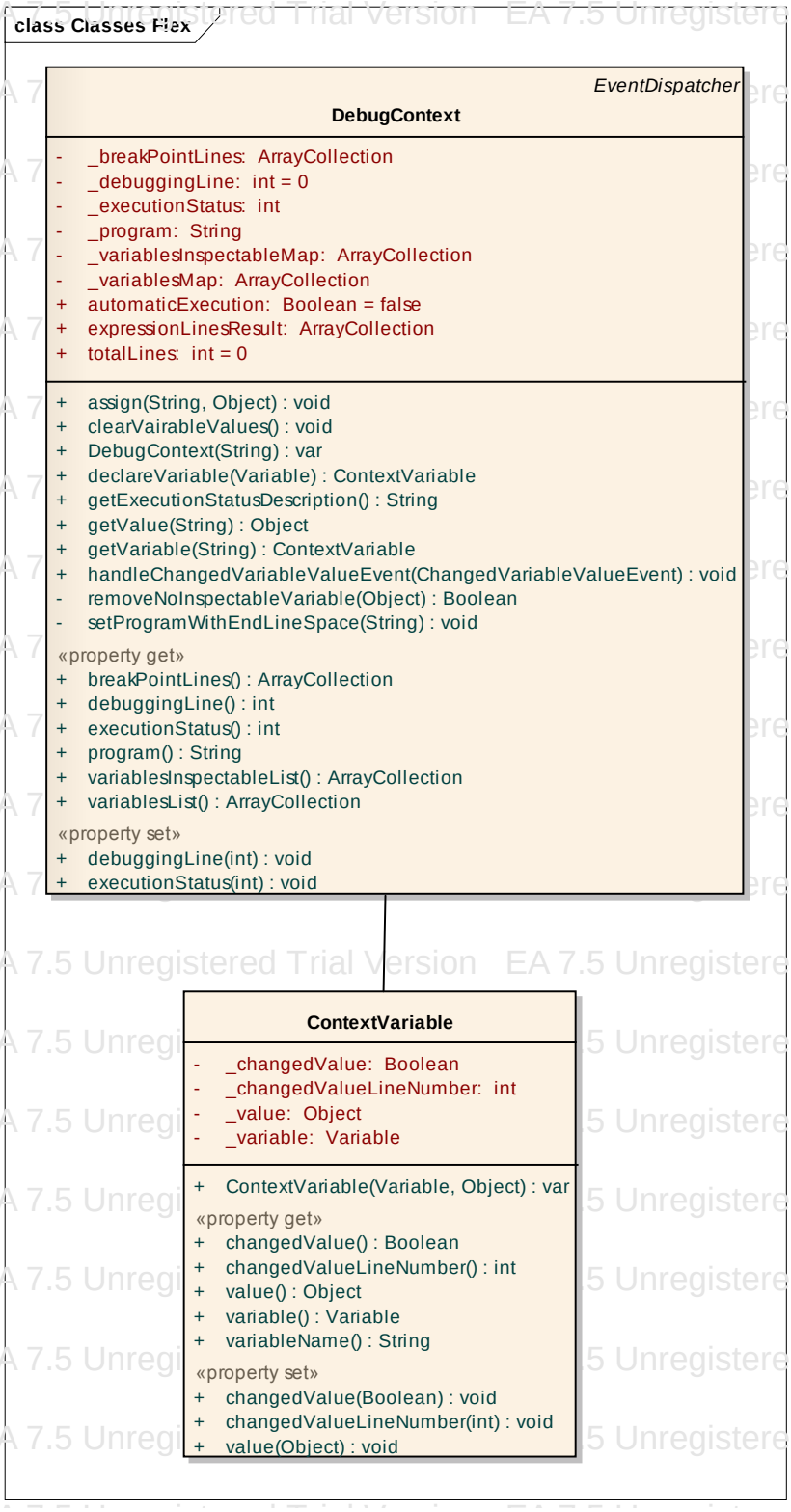


Figura 23 - Modelagem das classes que possuem informações do contexto de execução do algoritmo.

Na definição do padrão *Interpreter* temos outro componente, chamado de *Client*, este participante é responsável por as expressões e fazer as chamadas aos métodos *interpret* quando uma expressão deve ser executada [Gamma et al, 1995]. No projeto a classe *Client* é o *DebugController*, pois é a classe que controla todos os componentes do depurador de algoritmo.

No anexo 1 podemos ver a modelagem através de um diagrama de classes de todas as classes que compõem a implementação do padrão de projeto *Interpreter* neste trabalho.

Na camada Flex criamos também classes que são espelhos das classes que possuem as informações que são enviadas do Java para o Flex. Dessa forma todas as informações enviadas da camada Java para a Flex são recebidas após a requisição da mesma forma que foram montadas e estruturadas na camada Java. Podemos ver na Figura 24 a modelagem dessas classes.

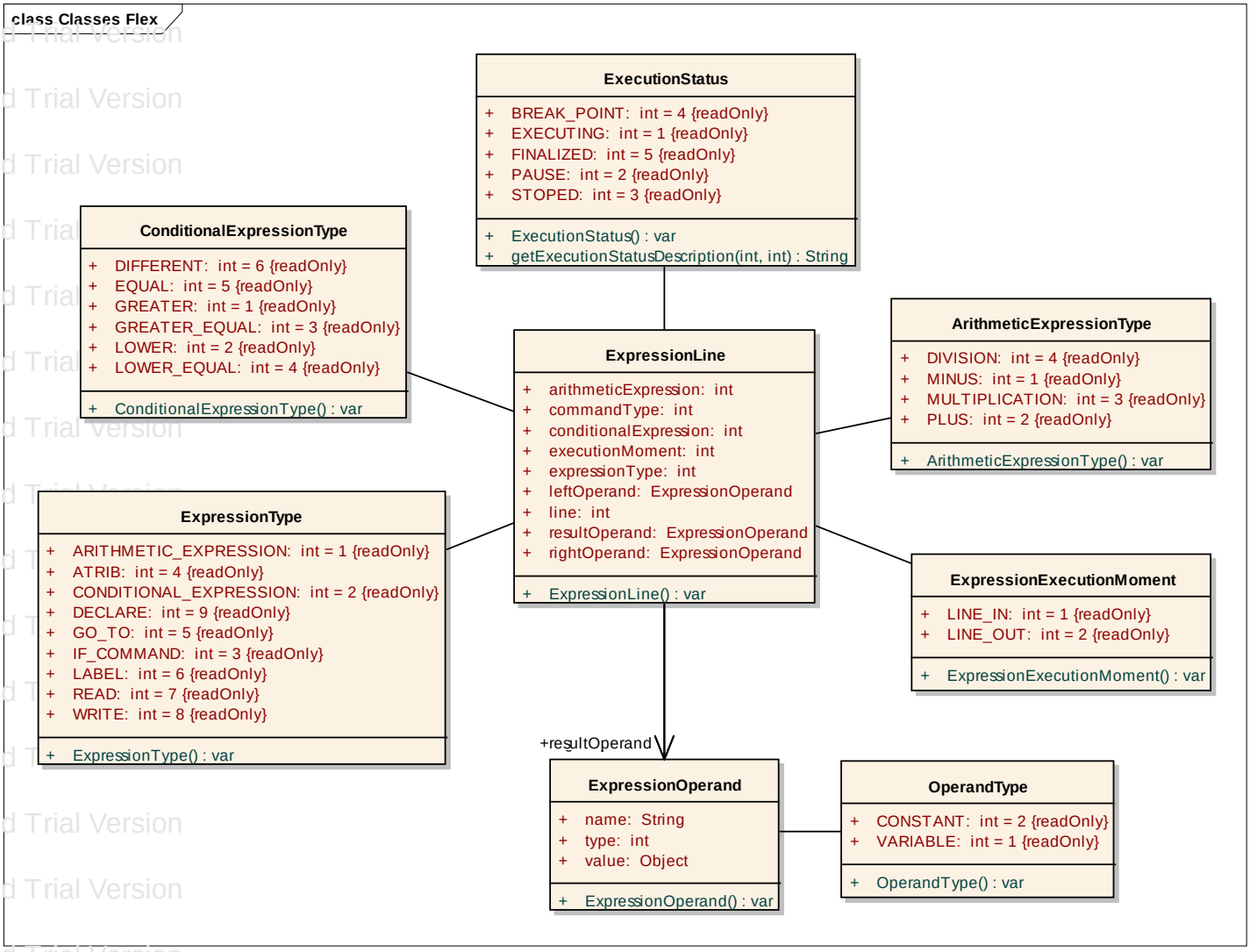


Figura 24 – Modelagem das classes espelhos que recebem as informações enviadas pela camada Java.

4.4 Cenários de Uso do Ambiente de Depuração

Para apresentarmos a ferramenta de depuração de algoritmos e como ela pode ser útil para encontrar erros no algoritmo desenvolvido, considerou-se seguinte problema algorítmico.

> *Uma escola necessita desenvolver um programa que calcule de forma automática a média aritmética das notas dos alunos. Para efetuar esta operação, o programa deverá solicitar o número de alunos da turma e o número de avaliações que foram realizadas na turma. Para cada aluno deverá ser mostrada a sua nota final. Após o cálculo das médias, o programa deverá mostrar o nome e a nota final dos alunos que obtiveram a maior e a menor média.*

Elaborou-se uma solução para este problema contendo erros de lógica a fim de ilustrar como a ferramenta pode ser útil para o estudante encontrá-los. Veja no Código 4 o algoritmo utilizado.

Código 4 - Algoritmo com erros utilizado para apresentação da ferramenta.

```
1   algoritmo
2      declare media_nota,media_turma,maior_media,menor_media: numerico
3      declare nota, nota_final,nota1,nota2,num_alunos,aux: numerico
4      declare nome, nome_maior,nome_menor: literal
5
6      para aux de 1 ate num_alunos faca
7
8         escreva "Numero de alunos" //Leitura de num_alunos deve ser fora do laço de
9   repetição
10        leia num_alunos
11        media_nota<-0
12
13        escreva "Nome"
14        leia nome
15
16        escreva "Informe 1a nota"
17        leia nota1
18
19        escreva "Informe 2a. nota "
20        leia nota2
21
22        nota_final <- (nota1+nota2) / 2
23
24        escreva "Aluno " + nome
25        escreva "Media " + nota_final
26
27       se (aux=0) entao      //A condição correta seria aux = 1
28          maior_media<-nota_final
29          nome_maior<-nome
30          menor_media<-nota_final
31          nome_menor<-nome
32        fim se
33
34        se (nota_final > maior_media) entao
35          maior_media<-nota_final
36          nome_maior<-nome
37        fim se
38
39        se (nota_final<menor_media) entao
40          menor_media<-nota_final
41          nome_menor<-nome
42        fim se
43
44       media_nota <- media_nota + 1 //O calculo correto é media_nota<-
45  media_nota+nota_final
46
```

```
47      fim para
48
49      media_turma<-media_nota/num_alunos
50
51      escreva "Media das notas da turma "  + media_turma
52
53      escreva "Aluno que obteve media menor foi "  + nome_menor
54      escreva "A menor nota final foi "  menor_media
55
56      escreva "Aluno que obteve media maior foi "  + nome_maior
57      escreva "A maior nota final foi " + maior_media
58
59  fim algoritmo
```

Os erros foram inseridos nas linhas 8, 27 e 44 do algoritmo.

4.4.1 Apresentação Ferramenta de Depuração

Após o aluno digitar o algoritmo ele pode solicitar a verificação léxica, sintática e semântica. Se o código não possuir nenhum erro de sintaxe, o aluno pode acessar a ferramenta de depuração clicando no botão “Depurar”, em destaque na Figura 25.

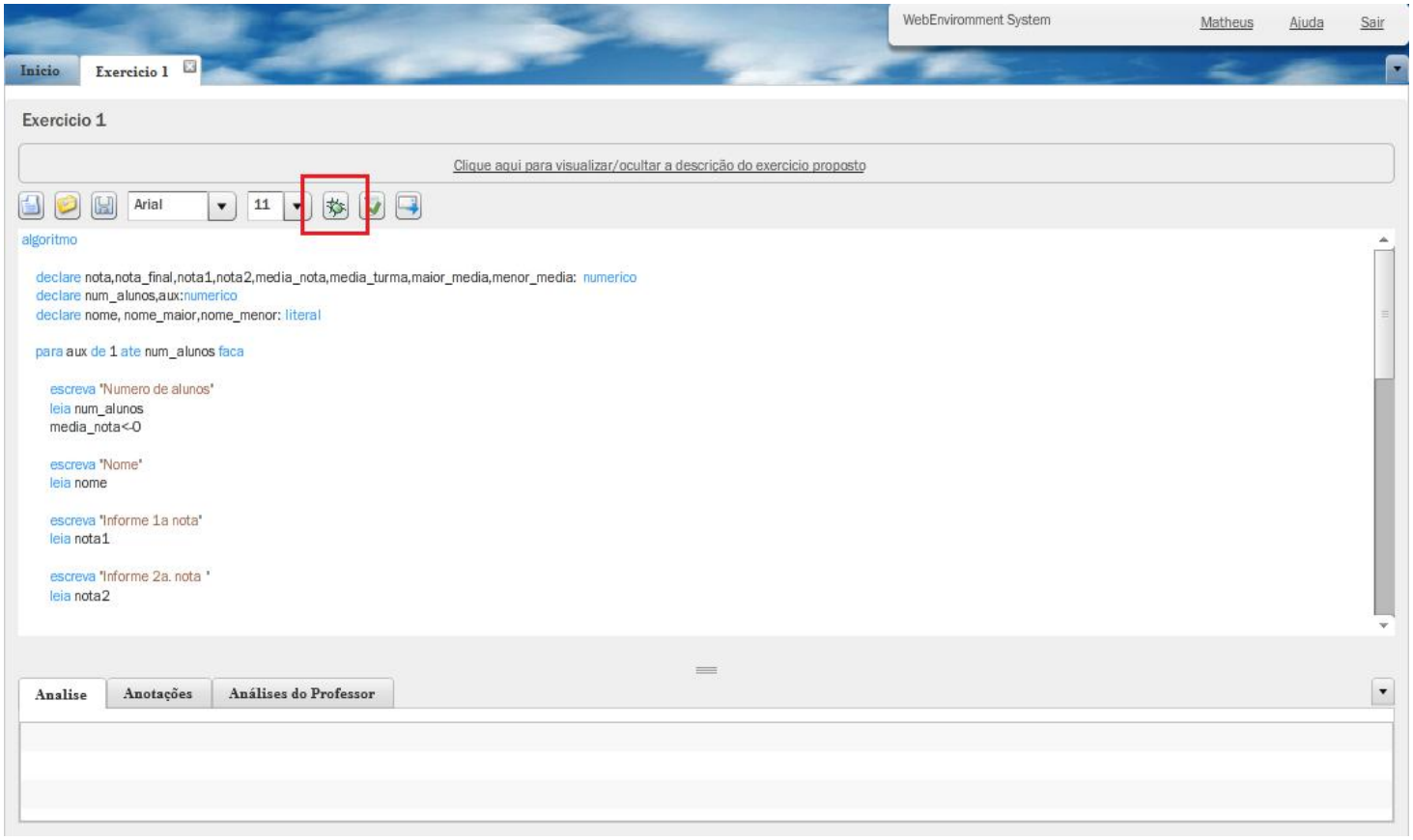


Figura 25 – Acesso ao depurador de algoritmos

Ao clicar no botão de depuração, o usuário é levado para outra tela aonde existem diversas ferramentas para realizar a depuração do algoritmo.

A tela principal do depurador é dividida em três grandes partes: uma janela com o algoritmo e a barra de ferramentas do depurador, a janela de inspeção de variáveis e a janela de mensagens de entrada e saída.

Na parte superior da interface observa-se uma barra de ferramentas para controle da execução e abaixo o algoritmo implementado. Ao lado direito da interface nota-se a janela de inspeção de variáveis, onde são apresentadas todas as variáveis definidas e seus valores. Na parte inferior tem-se uma janela de mensagens, onde são

apresentadas ao usuário as mensagens de saída e a interrupção para a entrada de valores. A Figura 26 ilustra a interface do depurador de algoritmos onde as janelas do depurador são destacadas.

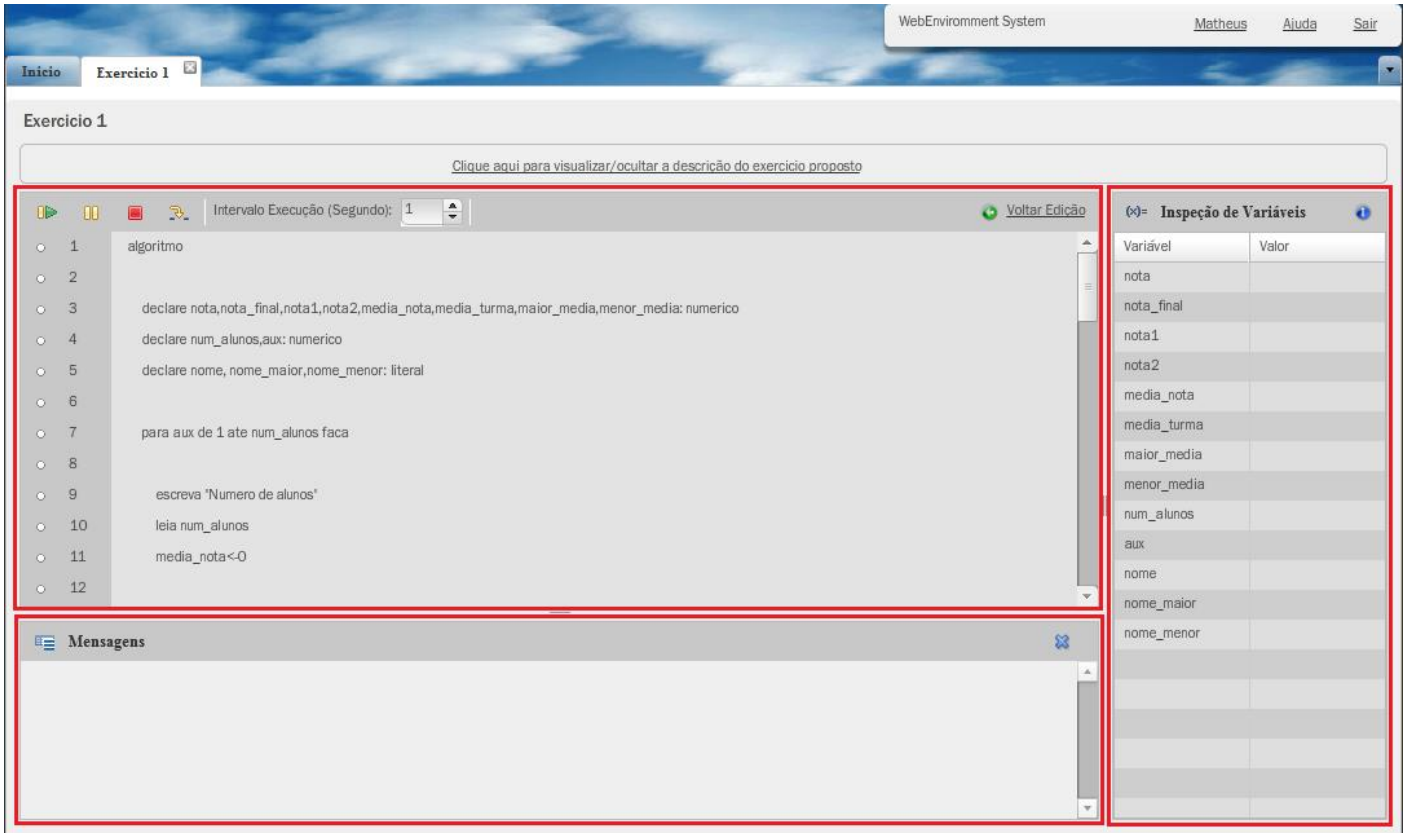


Figura 26 – Interface principal do depurador de algoritmo.

Quando o depurador é inicializado as janelas são abertas com um tamanho padrão, mas o usuário pode aumentar ou diminuir o tamanho de cada uma durante o processo de depuração.

4.4.2 Depuração de Algoritmo Passo a Passo

Após acessar a ferramenta de depuração o usuário pode executar o algoritmo passo-a-passo clicando no botão representado por uma seta na barra de ferramentas. A cada clique do botão, uma linha do código é executada.

A linha que está sendo executada é marcada com uma cor de fundo diferente das demais, para que o aluno a identifique facilmente.

No algoritmo implementado foi incluído um erro na leitura do número de alunos (linha 8), sendo que o correto seria que esta instrução estivesse fora do laço de repetição. Através da execução passo a passo o aluno pode verificar que se o fluxo de execução não entrar no laço de repetição, a variável não será lida.

Ao chegar à execução da linha seis (6), onde inicia o laço de repetição o usuário pode acompanhar o valor de cada variável e assim verificar que a variável “aux” possui o valor um (1) e a variável “num_alunos” não possui valor pois ainda não foi lida. Ao executar mais um passo do algoritmo a execução pula para o fim do laço de repetição, assim o estudante percebe que seu algoritmo está errado, pois ele deve conter a leitura do número de alunos antes do início do laço de repetição. A Figura 27 ilustra o caso apresentado.

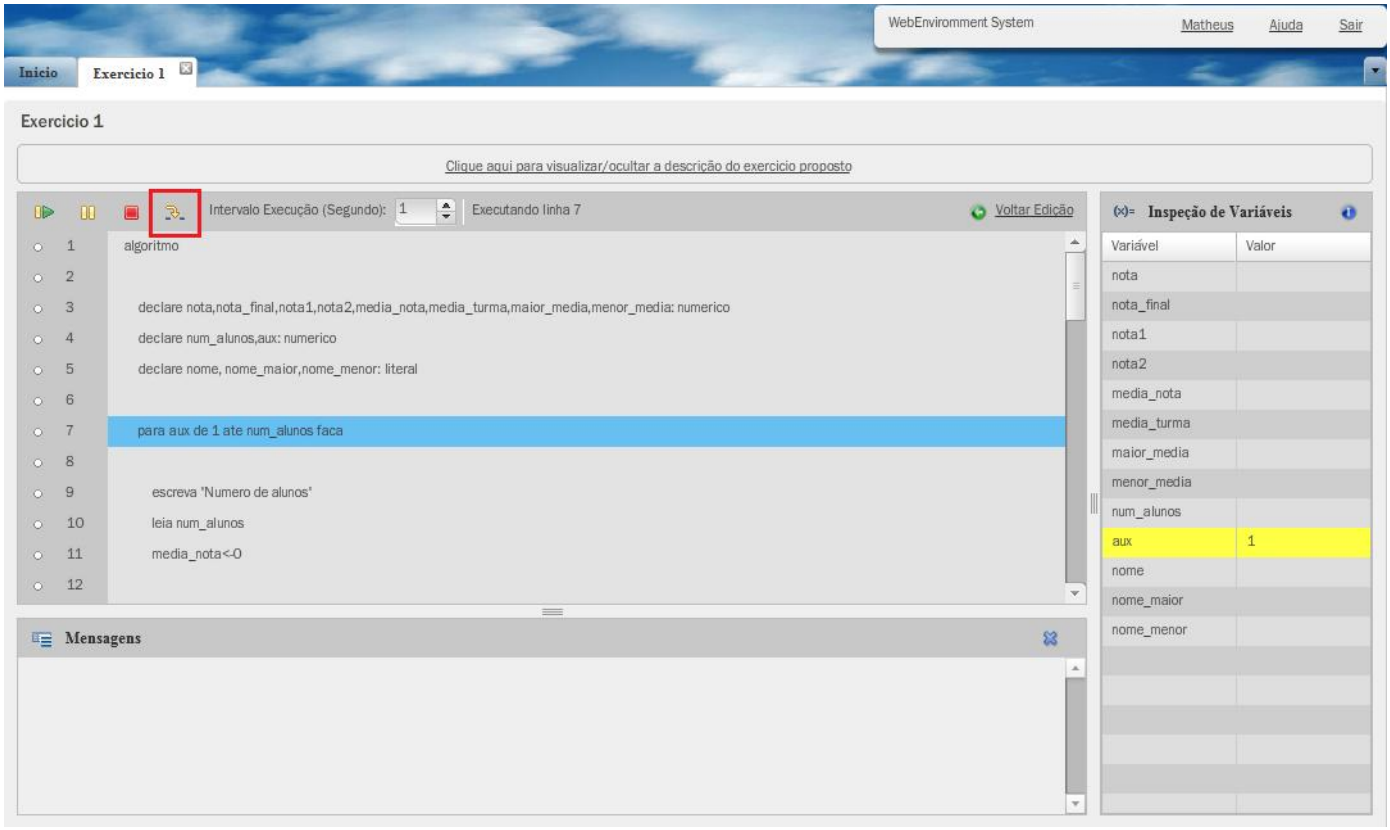


Figura 27 – Execução do algoritmo passo a passo.

Além da execução passo a passo o algoritmo pode ser executado direto do início ao fim, exceto se existir algum comando de leitura de variável. Neste caso, a ferramenta espera o usuário informar um valor para seguir a sua execução. O usuário também pode interromper uma execução direta clicando no botão de pausa existente na barra de ferramentas do depurador.

Existe também a possibilidade da inclusão de pontos de parada no código. Se o algoritmo estiver sendo executado diretamente e chegar a uma linha com um ponto de parada pré-definido, a execução será pausada até que o aluno continue a execução manualmente, através do botão de execução direta ou passo-a-passo.

Os pontos de parada podem se inseridos de duas formas: clicando sobre um ícone (representado por um círculo branco) existente ao lado do número de cada linha ou ainda clicando com o botão direito sobre a linha desejada, selecionando a opção de incluir ponto de parada. A remoção dos pontos de paradas pode ser feita da mesma forma que a sua inclusão (através de clique ou seleção). A Figura 28 destaca a execução de um algoritmo pausado por um ponto de parada inserido na linha quatro (4).

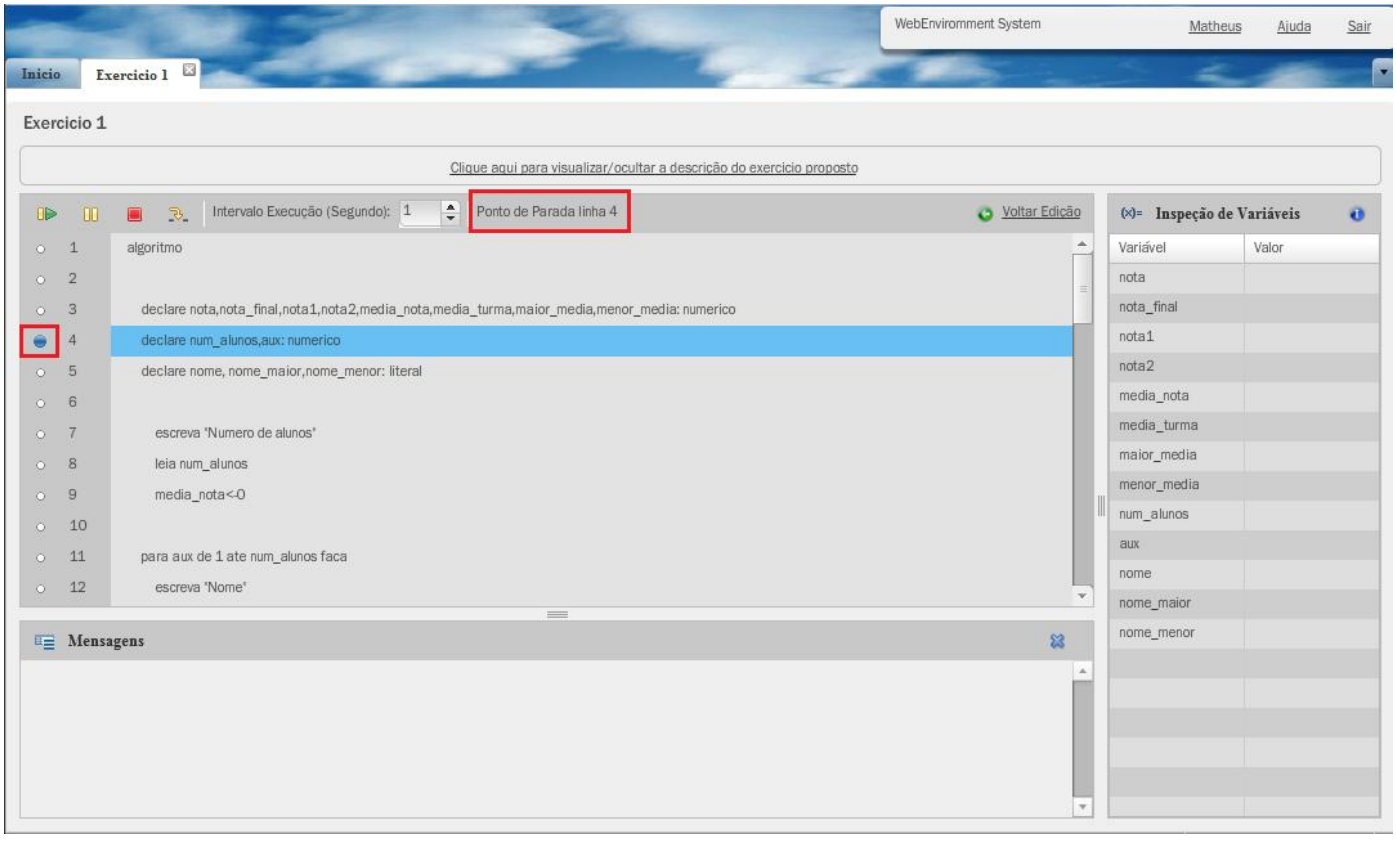


Figura 28 – Execução de uma linha com ponto de parada.

4.4.3 Escrevendo e Lendo Valores

Outra ferramenta existente no depurador de algoritmo é uma janela onde são apresentadas ao aluno todas as mensagens de entrada e saída do algoritmo.

Sempre que um comando escreva ou leia for executado, uma mensagem é apresentada na janela de Mensagens. Sempre que uma nova mensagem é apresentada ao usuário ela é incluída ao final da janela de mensagens, caso necessário, a barra de rolagem da janela é deslocada para o final do componente para que o usuário sempre visualize a última mensagem.

Mensagens de leitura interrompem a execução do algoritmo, a qual somente é reiniciada depois que um valor é informado. Quando um comando de leitura for executado uma caixa de texto será apresentada ao usuário no painel de mensagens para que o mesmo informe um valor.

Adicionalmente, a janela de mensagens possui um botão para o usuário limpar todas as mensagens já apresentadas na tela.

Veja na Figura 29 uma ilustração da janela de mensagens em destaque.

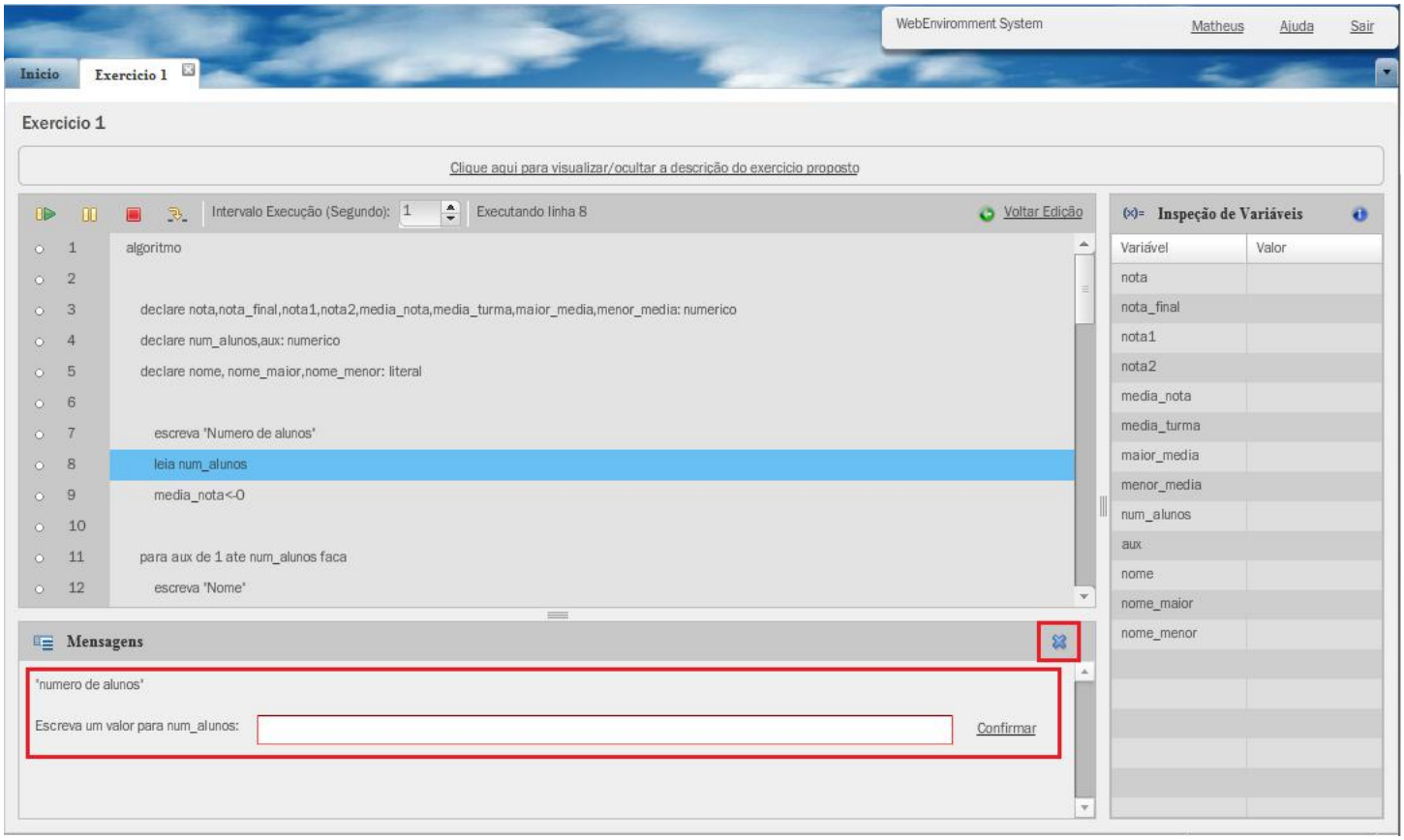


Figura 29 – Janela de mensagens de entrada e saída.

4.4.4 Inspecionando Variáveis

Continuando a resolução do algoritmo iniciado na seção 4.4.2, agora apresentaremos como encontrar erros no seu algoritmo utilizando a componente para inspeção de variáveis.

O componente utilizado para acompanhar o valor de cada variável durante a execução do algoritmo é localizado no lateral direita do depurador de algoritmos. Ao iniciar a depuração, todas as variáveis declaradas no algoritmo são identificadas no

componente de inspeção. Nesse momento nenhuma das variáveis possui ainda um valor.

Inserimos um erro no nosso algoritmo para que ele inicialize a variável de média apenas quando o valor da variável “aux” for igual a zero (0). Essa variável tem entretanto seu valor de inicialização igual a um (1). Ao executar o algoritmo passo-a-passo, quando chega-se na linha onde esta condição é verificada podemos visualizar na painel de inspeção de variáveis que o seu valor é igual a 1, logo a condição não será satisfeita. Se executarmos mais uma linha vamos comprovar que os comandos dentro da condição não serão executados. A Figura 30 ilustra claramente a etapa final do caso mencionado.

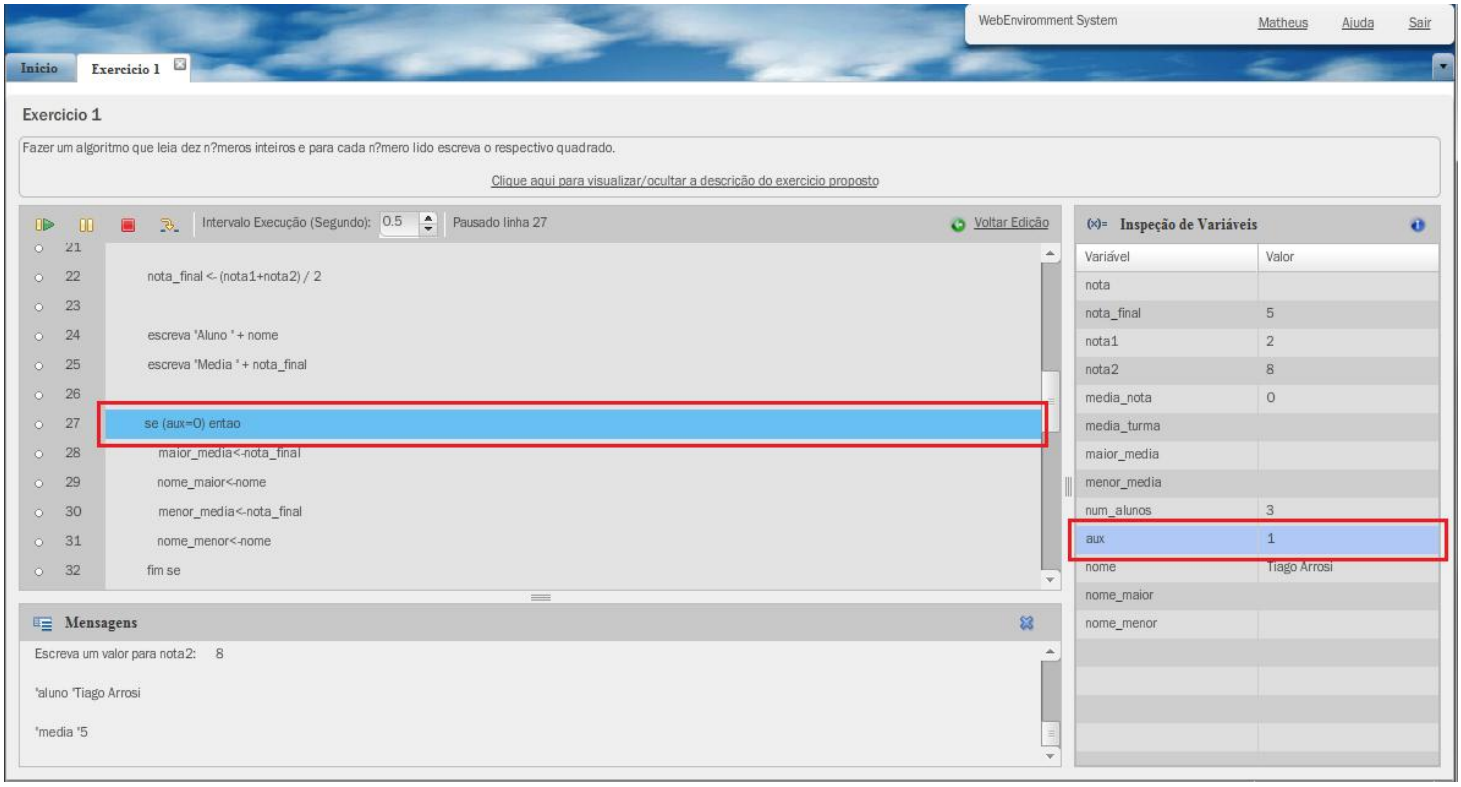


Figura 30 – Execução de comando condicional comparando com variável no painel de inspeção de variáveis.

Prossegue-se com a execução do algoritmo para realizar-se a correção do último erro incluído no algoritmo (linha 44). Esse erro se encontra na linha aonde são armazenadas as médias de cada aluno. O cálculo ao invés de incrementar a variável “media_nota” com o valor da variável “nota_final” ele está incrementando o valor em um (1).

Ao executarmos a linha aonde é feita a soma das médias de notas podemos acompanhar o valor da variável ser incrementada no painel de inspeção de variáveis. Sempre que o valor de uma variável é alterada, na execução de um comando, a linha

na listagem de variáveis fica em destaque mostrando ao usuário que algo aconteceu com aquela variável naquele momento.

Verificando o valor da variável no painel de inspeção de variáveis o estudante logo verá que a sua variável possui um valor incorreto e constatará que o seu algoritmo possui um erro na linha que está sendo executada. Assim o estudante não precisa verificar todo o seu algoritmo para encontrar aonde suas variáveis estão recebendo valores incorretos. A Figura 31 ilustra a alteração no valor da variável destacando a sua alteração no painel de inspeção.

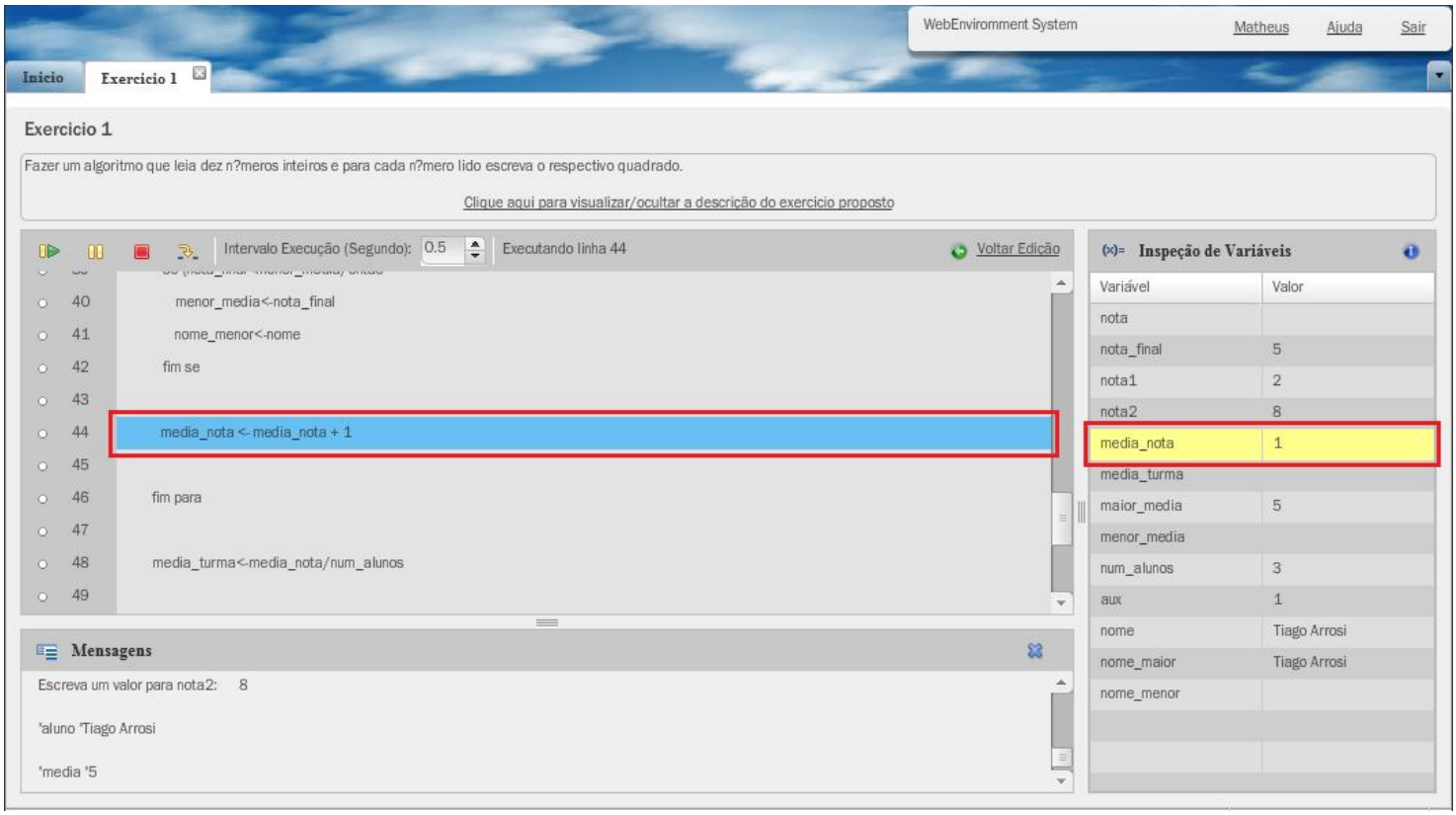


Figura 31 – Variável sendo alterada e destaque sendo apresentado no painel de inspeção de variáveis.

Além dessas funcionalidades apresentadas o através do painel de inspeção de variáveis podemos alterar o valor de qualquer variável em qualquer momento da execução do algoritmo.

Para alterar o valor de uma variável, basta que o aluno clique sobre a variável que deseja alterar no quadro de inspeção de variáveis. Ao clicar sobre a variável desejada, o campo se tornará editável e o usuário pode informar o valor desejado para a variável. Este valor será utilizado na variável durante a execução do restando do algoritmo, mas se a variável sofrer uma atribuição o seu valor será alterado normalmente.

No canto superior do painel de inspeção de variáveis foi incluído um ícone que fornece uma dica ao usuário, informando que ele pode alterar o valor de cada variável e como ele deve proceder para alterar elas. A Figura 32 ilustra a alteração de variáveis durante a execução do algoritmo.

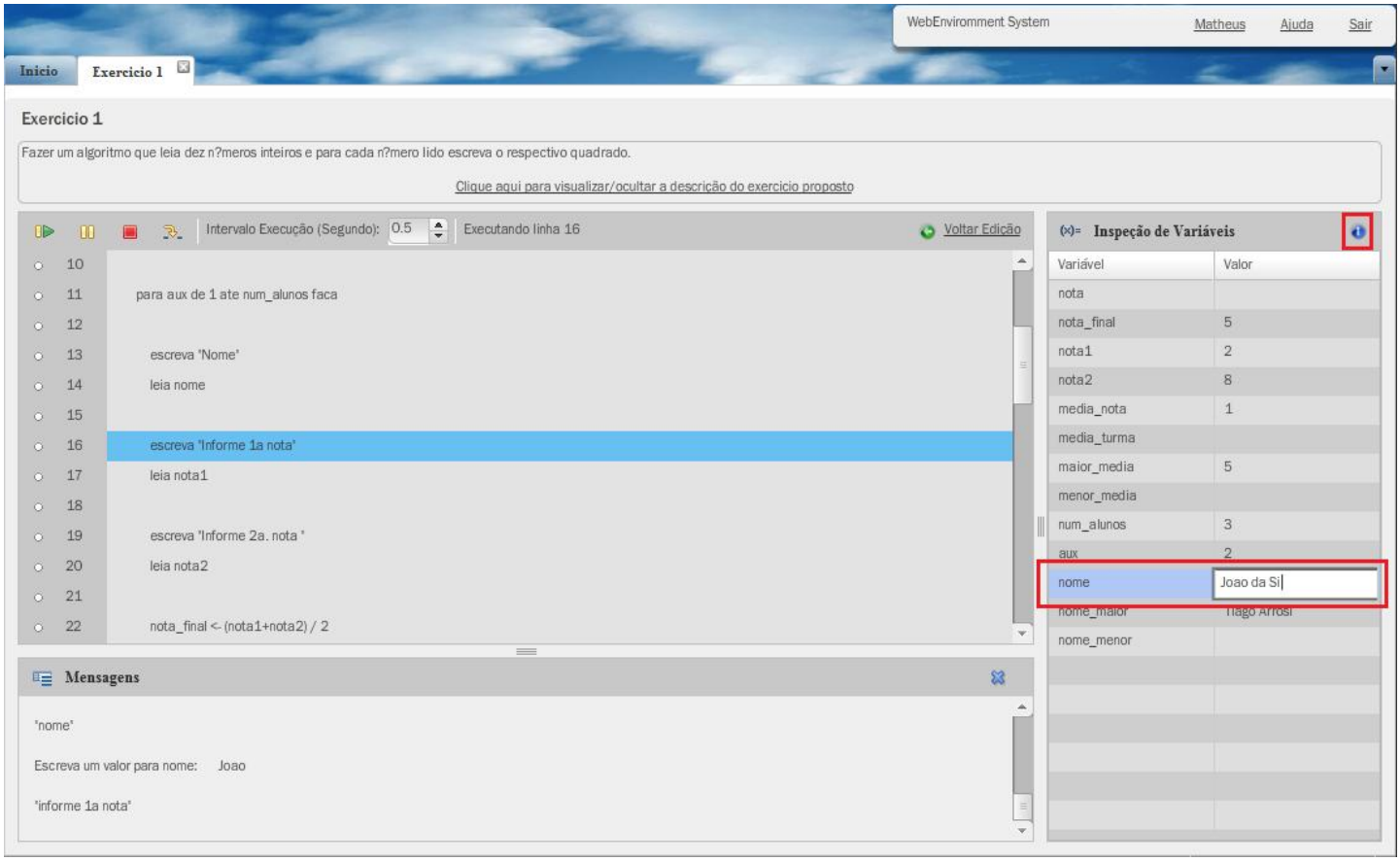


Figura 32 – Alteração do valor de uma variável durante a execução do algoritmo.

4.5 Avaliação do Ambiente de Aprendizagem

Com o objetivo de validar o presente trabalho foi realizado um teste com sete (7) alunos do curso de Licenciatura em Computação da Universidade de Caxias do Sul.

Foram escolhidos os estudantes de Licenciatura para realizar a avaliação do ambiente de aprendizagem, pois eles já possuem conhecimento para realizar uma avaliação pedagógica e técnica sobre um sistema. Consideramos que antes de realizar

um teste com alunos seria mais importante realizar um teste com pessoas capacitadas para testar e avaliar um sistema pedagógico.

A avaliação foi feita sobre o Ambiente de Aprendizagem desenvolvido no trabalho do Matheus Zenato [Zenato, 2009] e também sobre a ferramenta para depuração de algoritmos desenvolvida no presente trabalho.

Com esta avaliação pretendemos avaliar se o ambiente de aprendizagem e o depurador de algoritmos cumprem os objetivos propostos como: ajudar os alunos na resolução dos problemas, ajudar na aprendizagem de algoritmos, possuir uma navegação simples e fácil, controlar as ações dos estudantes e apontar as suas dificuldades no desenvolvimento dos algoritmos.

Para obtermos o resultado de cada avaliador individualmente utilizamos um formulário com perguntas pré-definidas, aonde cada aluno deveria preencher com a sua percepção do ambiente de aprendizagem e depurador de algoritmos. No formulário de avaliação existem perguntas que nos ajudam a verificar aonde o avaliador obteve mais dificuldade na utilização do sistema, o que pode ser melhorado para ajudar os estudantes na utilização, se as maiores dificuldades são em relação à utilização do sistema ou são relacionadas a questões didáticas e pedagógicas do ambiente de aprendizagem. O formulário da avaliação realizada pode ser consultada no Anexo 2 [Webber et al., 2009].

Iniciamos a avaliação apresentando brevemente o sistema e mostrando as funcionalidades existentes. Após a apresentação do sistema os alunos começaram a utilizá-lo e em seguida responderam uma avaliação sobre o Depurador de Algoritmos.

Os resultados obtidos em cada questão podem ser vistos na tabela 2, 3 e 4.

Tabela 2 – Resultado para pergunta sobre a classificação do ambiente de aprendizagem.

Avaliadores	**Classificação do Software:**
Avaliador 1	Exercício e Prática Ambiente de Aprendizagem Interativo(micromundos) Ambiente de Aprendizagem Cooperativo
Avaliador 2	Exercício e Prática
Avaliador 3	Exercício e Prática
Avaliador 4	Ambiente de Aprendizagem Cooperativo
Avaliador 5	Jogo Educativo Exercício e Prática Tutorial Ambiente de Aprendizagem Interativo (micromundos)
Avaliador 6	Sistema Tutor Inteligente Ambiente de Aprendizagem Interativo (micromundos) Ambiente de Aprendizagem Cooperativo
Avaliador 7	Exercício e Prática Tutorial

Tabela 3 – Resultado para pergunta sobre a atividade pedagógica apresentada pelo ambiente de aprendizagem.

Avaliadores	**Atividade pedagógica apresentada pelo software:**
Avaliador 1	Construtivista
Avaliador 2	Instrucionista/Comportamentalista
Avaliador 3	Instrucionista/Comportamentalista
Avaliador 4	Construtivista
Avaliador 5	Construtivista Sócio-interacionista
Avaliador 6	Construtivista
Avaliador 7	Sócio-interacionista

Tabela 4 – Resultado Perguntas Pedagógicas e Técnicas.

Perguntas	Avaliador 1	Avaliador 2	Avaliador 3	Avaliador 4	Avaliador 5	Avaliador 6	Avaliador 7
1	Parcialmente	Parcialmente	Parcialmente	Sim	Sim	Parcialmente	Parcialmente
2	Sim	Não se Aplica	Não se Aplica	Sim	Sim	Parcialmente	Parcialmente
3	Parcialmente	Não	Não	Não	Sim	Não	Sim
4	Parcialmente	Não	Não	Não	Sim	Não	Não
5	Não	Não	Não	Parcialmente	Sim	Não	Sim
6	Sim	Sim	Sim	Sim	Sim	Parcialmente	Sim
7	Sim	Sim	Sim	Sim	Parcialmente	Parcialmente	Não
8	Não se Aplica	Não se Aplica	Não se Aplica	Sim	Parcialmente	Sim	Não
9	Sim	Parcialmente	Parcialmente	Sim	Parcialmente	Sim	Parcialmente
10	Não	Não	Não	Não	Não	Não	Não
11	Não se aplica	Não se Aplica	Não se Aplica	Sim	Não se Aplica	Sim	Não se Aplica
12	Parcialmente	Sim	Sim	Sim	Parcialmente	Parcialmente	Parcialmente
13	Não se Aplica	Não se Aplica	Não se Aplica	Não se Aplica	Não se Aplica	Parcialmente	Não se Aplica
14	Sim	Não se Aplica	Não se Aplica	Sim	Parcialmente	Sim	Sim
15	Sim	Parcialmente	Parcialmente	Sim	Parcialmente	Sim	Parcialmente
16	Sim	Sim	Sim	Sim	Sim	Sim	Não se Aplica
17	Não	Não	Não	Parcialmente	Parcialmente	Não	Não
18	Sim	Não se Aplica	Não se Aplica	Sim	Sim	Sim	Sim
19	Sim	Não se Aplica	Não se Aplica	Sim	Sim	Sim	Sim
20	Não	Sim	Sim	Não se Aplica	Não	Não	Não
21	Não	Não se Aplica	Não se Aplica	Não	Não	Não	Não se Aplica
22	Não	Não se Aplica	Não se Aplica	Não se Aplica	Não se aplica	Não	Não se Aplica
23	Não	Não se Aplica	Não se Aplica	Não	Não	Não	Não
24	Sim	Parcialmente	Parcialmente	Sim	Parcialmente	Parcialmente	Não

Analisando os resultados podemos chegar a algumas conclusões sobre o ambiente de aprendizagem. Todos os alunos classificaram o sistema como um Ambiente de Aprendizagem Cooperativo e Interativo ou como um sistema de Exercícios e Prática.

Outro ponto importante constatado foi que todos os alunos avaliadores disseram que o sistema é adequado totalmente ou parcialmente ao nível do seu público-alvo. Em relação à variedade de atividades e níveis de complexidade das tarefas os estudantes confirmaram que o sistema possui suporte, mas que isso depende dos professores que serão os responsáveis por inserir no sistema as atividades que devem ser realizadas.

Em relação aos aspectos técnicos do ambiente de aprendizagem os alunos avaliadores entendem que o usuário possui controle sobre as suas atividades de forma a interrompê-las e retomá-las quando desejado.

Outra avaliação positiva ao trabalho é que o ambiente de aprendizagem possui uma interface agradável e que estimula a sua utilização.

Um ponto que foi sentido falta pelos avaliadores é a inexistência de menus de ajuda ou dicas que possam ajudar o usuário na utilização do sistema e também na resolução dos problemas.

Os avaliadores consideraram que o ambiente de aprendizagem pode ser utilizado pelos alunos sem a interferência do professor. Esse resultado é muito importante, pois um dos propósitos do sistema é que o aluno possa acessá-lo de

qualquer lugar, sem a presença de um tutor e que consiga utilizar o sistema sem encontrar dificuldades que o desanime e desmotive a aprender.

Uma sugestão de melhoria para o trabalho, sugerida por um aluno, seria de aproximar a linguagem de algoritmos do trabalho com a linguagem do sistema Visualg [Souza, 2009] e incluir um menu de ajuda com a sintaxe aceita pelo compilador do sistema.

Durante a avaliação do ambiente de aprendizagem o sistema se comportou muito bem, apenas um erro foi encontrado pelos alunos. Quando foi solicitada a avaliação do algoritmo pelo SIATP, ocorreu um erro e o sistema não obteve uma resposta do avaliador. Esse erro acabou gerando outros erros no ambiente de aprendizagem, por isso tivemos que sair da aplicação e entrar novamente para continuar a avaliação.

5 Conclusão

5.1 Síntese

Este trabalho teve como um dos objetivos estudar ambientes de aprendizagem baseados na resolução de problemas, buscando suas características, arquitetura interna de componentes e a busca por soluções utilizadas que estão dando certo nesses ambientes de aprendizagem. Após entender os ambientes de aprendizagem foram realizados testes e análises sobre alguns ambientes utilizados na programação.

Com base no estudo sobre ambientes de aprendizagem e das análises sobre alguns ambientes de programação foram levantados alguns requisitos que para ajudar alunos a encontrar erros em algoritmos e também a entendê-los. Foram levantados três requisitos para serem implementados nesse trabalho: Depuração de código, Testes automatizados e Fluxograma do Algoritmo.

Com os requisitos levantados foi realizada a definição de arquitetura do projeto e as tecnologias que serão utilizadas no desenvolvimento do ambiente de aprendizagem. Com isso foi possível iniciar a modelagem das classes e iniciar a sua implementação.

Assim, neste trabalho foi implementado o depurador de algoritmo com inúmeras funcionalidades acopladas que podem ajudar o estudante a encontrar os pontos no seu algoritmo aonde existem erros de implementação.

5.2 Contribuições do Trabalho

Através do estudo sobre ambientes de aprendizagem para áreas complexas, definiram-se três ferramentas úteis para o aprendizado e o auxílio na verificação de erros em algoritmos, são elas: Depurador de Algoritmo, Execução de testes automatizados e Fluxograma do Algoritmo. Dentre essas ferramentas implementou-se o depurador de algoritmo pois entendemos ser a ferramenta mais completa dentre as levantadas, pois ajuda o aluno a visualizar o fluxo que o seu algoritmo segue, acompanhar o valor de cada variável durante a sua execução e também auxiliar no verificação de erros.

Este trabalho trouxe diversas possibilidades e melhoria no aprendizado de algoritmos, agora o aluno pode realizar todas as suas tarefas dentro de um ambiente de aprendizagem e o professor pode acompanhar os resultados e dificuldades do aluno.

Com este trabalho espera-se também que novas ferramentas para aprendizagem de algoritmos sejam criadas, focando principalmente no aprendizado de algoritmos e que possam ser incluídas dentro de ambientes virtuais de aprendizagem.

5.3 Perspectivas Trabalhos Futuros

Este trabalho apresentou três ferramentas que podem ajudar e auxiliar os estudantes durante o seu aprendizado em algoritmos. Essas ferramentas foram

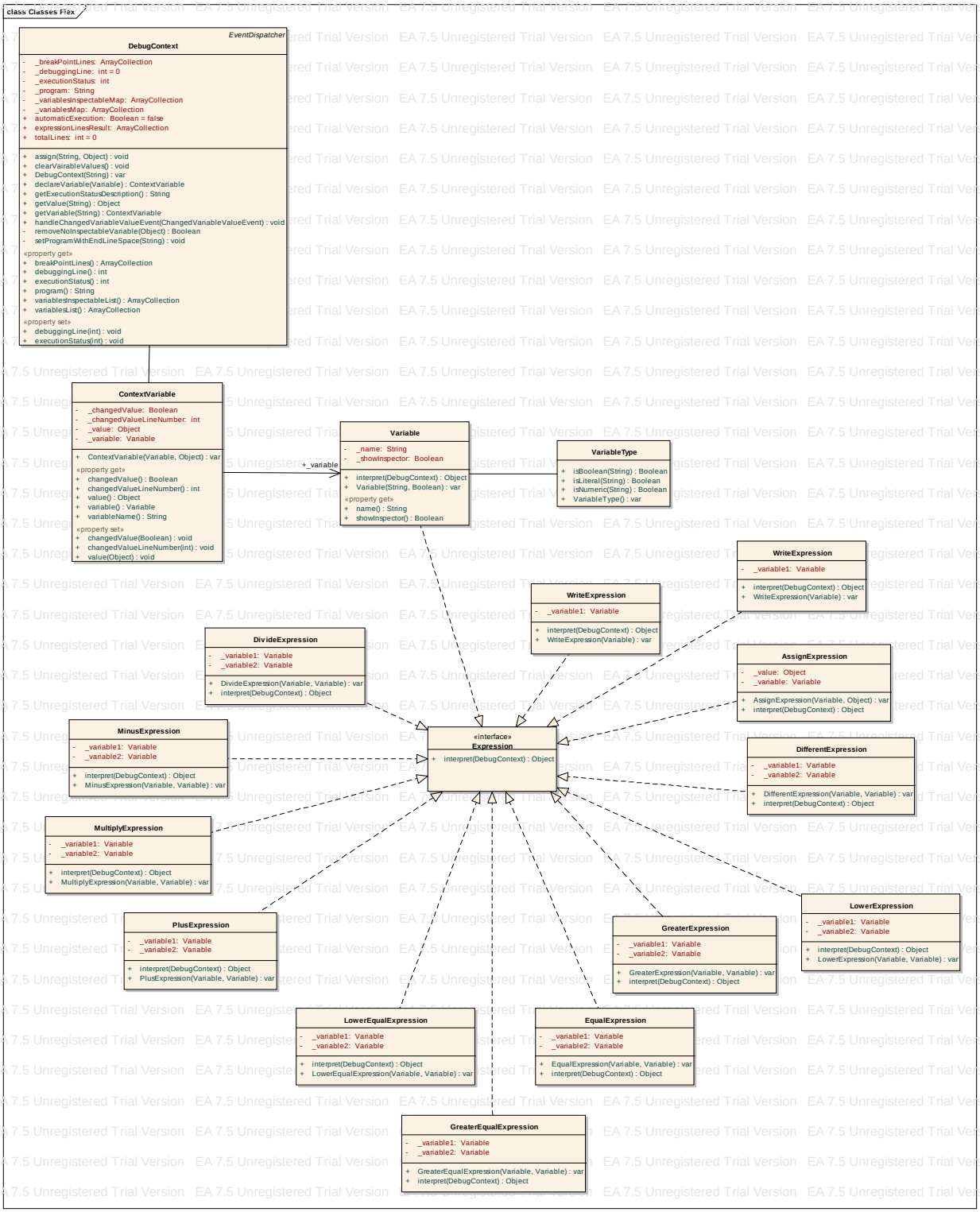
definidas através do estudo e testes sobre ambientes de aprendizagem e ferramentas profissionais utilizadas no desenvolvimento de algoritmo. Ainda existe um leque aberto de ferramentas que podem ser criadas e acopladas ao sistema de aprendizagem, facilitando ainda mais o aprendizado do aluno.

Outro ponto que pode ser aprofundado é na própria ferramenta de depuração, onde diversas funcionalidades podem ser analisadas e incluídas. Conforme sugestões recebidas durante a avaliação do depurador, ele carece de menus e sistemas de ajuda que possam ajudar o aluno no aprendizado e também na utilização da ferramenta.

Analisando o resultado da avaliação realizada sobre o ambiente de aprendizagem encontramos mais alguns pontos que podem ser melhorados como a inclusão de manuais no ambiente de aprendizagem, menus de ajuda para o estudante e a atualização da linguagem utilizada pelo compilado, aproximando mais com outras ferramentas também utilizadas para aprendizagem de algoritmos.

6 ANEXOS

Anexo 1 - Modelagem das classes que compõem a implementação do padrão Interpreter.



Anexo 2 – Formulário para avaliação do Depurador de Algoritmos

Universidade de Caxias do Sul

Centro de Computação e Tecnologia da Informação

Avaliação de Software Educativo

Este instrumento foi aplicado para avaliar o *Software Caderno de Algoritmos* desenvolvido pelos alunos Tiago Arrosi e Matheus Zenato na disciplina de TCC do curso de Ciência da Computação.

Nome do Avaliador:

Formação:

I - Classificação do Software:

( ) Jogo educativo
( ) Simulador
( ) Exercício e prática
( ) Tutorial
( ) Hipermídia
( ) Sistema Tutor Inteligente
( ) Ambiente de aprendizagem interativo (micromundos)
( ) Sistema de Autoria
( ) Ambiente de aprendizagem cooperativo

II - Atividade pedagógica apresentada pelo software:

( ) Instrucionista/Comportamentalista

( ) Construtivista

( ) Sócio-interacionista

ETAPA A - ASPECTOS PEDAGÓGICOS	
01	O software propõe situações-problema que envolvam a formulação de hipóteses, a investigação e/ou a comparação? ( ) Sim ( ) Parcialmente ( ) Não ( ) Não se aplica

02	É adequado ao nível do aprendiz (público-alvo)? ( ) Sim ( ) Parcialmente ( ) Não ( ) Não se aplica
03	Instiga a procura de outras informações em diferentes fontes de pesquisa? ( ) Sim ( ) Parcialmente ( ) Não ( ) Não se aplica
04	Ele favorece a utilização interdisciplinar? ( ) Sim ( ) Parcialmente ( ) Não ( ) Não se aplica
05	O software apresenta atividades variadas, variando os níveis de complexidade das tarefas? ( ) Sim ( ) Parcialmente ( ) Não ( ) Não se aplica

ETAPA B - ASPECTOS TÉCNICOS	
06	O usuário tem controle do software, podendo interromper as tarefas, retomá-las, corrigir erros? ( ) Sim ( ) Parcialmente ( ) Não ( ) Não se aplica
07	A apresentação de textos é adequada à leitura (tamanho da fonte, cor das fontes utilizadas)? ( ) Sim ( ) Parcialmente ( ) Não ( ) Não se aplica
08	As imagens auxiliam na compreensão dos conteúdos, sendo mencionadas nos textos explicativos? Elas não são apenas utilizadas de maneira decorativa? Por exemplo, ao tratar o assunto de funcionamento do motor, o software apresenta uma imagem de um tipo de motor. ( ) Sim ( ) Parcialmente ( ) Não ( ) Não se aplica

09	Os grafismos e layout de tela são bonitos, estimulando a utilização do software? ( ) Sim ( ) Parcialmente ( ) Não ( ) Não se aplica
10	O software dispõe de helps (ajuda) ou dicas? ( ) Sim ( ) Parcialmente ( ) Não ( ) Não se aplica
11	Recursos de animação são bem empregados (complementam o conteúdo, e não apenas distraem o usuário e decoram o sistema)? ( ) Sim ( ) Parcialmente ( ) Não ( ) Não se aplica
12	As instruções do software são claras, indicando de maneira precisa o que deve ser realizado? ( ) Sim ( ) Parcialmente ( ) Não ( ) Não se aplica
13	O programa opera de acordo com as instruções fornecidas na documentação? ( ) Sim ( ) Parcialmente ( ) Não ( ) Não se aplica
14	A linguagem apresentada é adequada? (Adequada ao público a que o software se destina, adequada ao tópico/conteúdo, características regionais) ( ) Sim ( ) Parcialmente ( ) Não ( ) Não se aplica
15	As convenções e símbolos utilizados para guiar o usuário na interface são usuais? ( ) Sim ( ) Parcialmente ( ) Não ( ) Não se aplica
16	Quando um erro ocorre, o sistema consegue contorná-lo, permitindo que o usuário cancele a operação ou retorne a um ponto anterior? ( ) Sim ( ) Parcialmente ( ) Não ( ) Não se aplica
17	Utiliza recursos multimídia (textos, imagens, sons, filmes) para apresentar os assuntos? ( ) Sim ( ) Parcialmente ( ) Não ( ) Não se aplica

18	O vocabulário é adequado ao público-alvo? ( ) Sim ( ) Parcialmente ( ) Não ( ) Não se aplica
19	O conteúdo é apropriado ao nível do aluno? ( ) Sim ( ) Parcialmente ( ) Não ( ) Não se aplica
20	O software observa a correção da ortografia? ( ) Sim ( ) Parcialmente ( ) Não ( ) Não se aplica
21	O software apresenta manual do usuário? ( ) Sim ( ) Parcialmente ( ) Não ( ) Não se aplica
22	O software apresenta manual de instalação? ( ) Sim ( ) Parcialmente ( ) Não ( ) Não se aplica
23	O software apresenta manual pedagógico ao professor? ( ) Sim ( ) Parcialmente ( ) Não ( ) Não se aplica
24	O aluno consegue utilizar o software sem a interferência do professor? ( ) Sim ( ) Parcialmente ( ) Não ( ) Não se aplica

Você tem comentários ou sugestões para o aprimoramento do software? Use o espaço abaixo.

7 BIBLIOGRAFIA

Adobe Systems, Inc. **Adobe Flex.** Disponível em http://www.adobe.com/products/flex/?promoid=BPDEQ, acessado em 05/06/2009.

Almeida, M. E. B. **Educação a distância na internet: abordagens e contribuições dos ambientes digitais de aprendizagem**, São Paulo, Revista Educação e Pesquisa, 29(2):327-340. 2003.

Alur, D., Crupi, J., & Malks, D. (2004). Core J2EE patterns: as melhores práticas e estratégias de design, Rio de Janeiro, Elsevier.

Ardito, C., De Marsico, M., Lanzilotti, R., Levialdi, S., Roselli, T., Rossano, V., & Tersigli, M. (2004). **Usability of E-Learning Tools**. Proceedings of AVI 2004, May 25-28, 2004, Gallipoli, Italy, 80-84.

Arnow, D. and Barshay, O. 1999. **WebToTeach: An Interactive Focused Programming Exercise System**. In *Proceedings of Frontiers in Education Conference*, San Juan, Puerto Rico, November 1999, Session 12a9.

Arsand, Eduardo André. **Implementação de interfaces auto-adaptáveis para um ambiente de aprendizagem**. 2005. Trabalho de Conclusão de Curso. (Graduação em Bacharelado em Ciência da Computação) - Universidade de Caxias do Sul, Caxias do Sul, RS.

Baldwin, D. 1996. **Three years experience with Gateway Labs**. In Proceedings of Innovation and Technology in Computer Science Education Conference, Barcelona, Spain, June 1996, ACM Press, 6-7.

Bottentui Junior, J. B.; Coutinho, C.P. (2008) **As Ferramentas da Web 2.0 no apoio à Tutoria na Formação em E-learning**. In XVI Colóquio da Association Francophone Internationale de Recherche Scientifique en Education - (AFIRSE), Lisboa, Portugal.

Bridgeman, S., Goodrich, M.T., Kobourov, S.G., and Tamassia, R. 2000. **PILOT: An Interactive Tool for Learning and Grading**. In Proceedings of the SIGCSE Technical Symposium on Computer Science Education, Austin, TX, March 2000, ACM Press, 139-143.

Brusilovsky P., **The Intelligent Tutor, Environment and Manual for Introductory Programming**. Educational Technology and Training International, 1992, v.29, n.1, p.26-34.

Cavalcante, R., Finley T., and Rodger, S.H. 2004. **A Visual and Interactive Automata Theory Course with JFLAP 4.0**, In Proceedings of the 35th SIGCSE Technical Symposium on Computer Science Education, Norfolk, VA, March 2004, 140-144.

K. E. Chang, Y. T. Sung, and S. F. Lin, (2006), **Computer-assisted learning for mathematical problem solving**, Computer and Education: An international Journal, vol. 46/2, 140-151. (SSCI, SCI)

Costa, L. A. C. e Franco, S. R.K. **Ambientes Virtuais De Aprendizagem e suas Possibilidades Construtivistas**. REVISTA NOVAS TECNOLOGIAS NA EDUCAÇÃO, v. 3, n. 1. Maio, 2005.

Delamaro, Marcio. **Como Construir um Compilador Utilizando Ferramentas Java**. Novatec, 2004. ISBN 85-7522-055-1.

Eclipse Foundation. **Eclipse.** Disponível em http://www.eclipse.org/, acessado em 18/05/2009.

Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides, **Design patterns: elements of reusable object-oriented software**, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, 1995.

Franco, Marcelo Araújo; Cordeiro, Luciana Meneghel and CASTILLO, Renata A. Fonseca del. **O ambiente virtual de aprendizagem e sua incorporação na Unicamp.** *Educ. Pesqui.* [online]. 2003, vol.29, n.2 ISSN 1517-9702. Similarity:0.391704.

Garner, S., **Learning Resources and Tools to Aid Novices Learn Programming**. in Informing Science & Information Technology Education Joint Conference (INSITE), (Pori, Finland, 2003), 213--222.

Gertner, A. & VanLehn, K.(2000) **Andes: A Coached Problem Solving Environment for Physics**. In G. Gauthier, C. Frasson & K. VanLehn (Eds), *Intelligent*

Tutoring Systems: 5th International Conference. Berlin: Springer (Lecture Notes in Computer Science, Vol. 1839), pp. 133-142.

Gomes, A.; Mends, A., **Learning to program - difficulties and solutions**, 10th International Conference on Engineering Education, Session F1B, p. 283, September 2007.

Gomes, A.; Carmo L.; Bigotte, E.; Mendes, A. , **"Mathematics and programming problem solving"**, 3rd E-Learning Conference – Computer Science Education, Coimbra, September 2006.

Joseph Beck, Mia Stern, and Erik Haugsjaa, **Applications of AI in Education**, ACM Crossroads, September, 1996.

Kumar, A.N., **Explanation of step-by-step execution as feedback for problems on program analysis, and its generation in model-based problem-solving tutors**, Technology, Instruction, Cognition and Learning (TICL) Journal, Special Issue on Problem Solving Support in Intelligent Tutoring Systems.

Kumar, A.: **Generation of problems, answers, grade, and feedback case study of a fully automated tutor**. 5(3) (September 2005) Article 3.

Martínez-Santaolalla, M.J; Bienvenida, B.F; Túnez, R. S. **ICT In Mathematics Education: Geometry Problem Solving With Applets**, Recent Research Developments In Learning Technologies, Department Of Languages And Computation, University of Almería, La Cañada De San Urbano. 2005.

Mitrovic, A., Ohlsson, S., Martin, B. (2006) **Problem-solving support in constraint-based tutors**. *Technology, Instruction, Cognition and Learning. Special issue* on Highlights on AERA 2005, vol 3, no 1-2, pp. 43-50, 2006.

Mota, Marcelle P.; Pereira, Lis W.K.; Favero, Eloi L. **JavaTool: Uma ferramenta para ensino de programação.** Workshop sobre Educação em Computação - WEI - Belém - 2008.

Neto, J. **Reflexão em torno do Magalhães**. Disponível em http://www.imagina.pt/index.php?option=com_content&view=article&id=693%3Areflexao-em-torno-do-magalhaes&catid=353%3Aquestoes-e-aprendizagem&Itemid=706&lang=pt, acessado em 05/05/2009.

Nicaud, J.F, Bouhineau, D., Huguet, T., 2002. **The Aplusix-Editor: A new kind of software for the learning of algebra**. Intelligent Tutoring Systems, ITS 2002. Lectures Notes in Computer Science, 2363:178-187.

Pasinatto, E. **Concepção de interfaces auto-adaptáveis para ambientes de aprendizagem.** 2005. Trabalho de Conclusão de Curso. (Graduação em Bacharelado em Ciência da Computação) - Universidade de Caxias do Sul, Caxias do Sul, RS.

Possamai, R. **Modelagem de um Sistema de Diagnóstico do Aluno Inspirado na Teoria do Perigo**. 2008. Trabalho de Conclusão de Curso. (Graduação em Bacharelado em Ciência da Computação) - Universidade de Caxias do Sul, Caxias do Sul, RS.

Schulze, K.G., Shelby, R.N., Treacy, D.J., Wintersgill, M.C. (2000). **Andes: A Coached Learning Environment for Classical Newtonian Physics.** In *Proceedings of the 11th International Conference on College Teaching and Learning*. Jacksonville, FL, April, 2000.

Scott A, Watkins M and McPhee D,*2008*, **E-Learning For Novice Programmers - A Dynamic Visualisation and Problem Solving Tool**, Proceedings of the ICTTA 2008, *Syrian Computing Society / IEEE*, Damascus – Syria

Scott, Andrew. **Programinate**. Disponível em http://www.comp.glam.ac.uk/pages/staff/asscott/programinate/index.html, acessado em 18/05/2009.

Sguillaro, Daniel. **Um Compilador Em Java para Fins Didáticos**. 2006. Trabalho de Conclusão de Curso. (Graduação em Bacharelado em Ciência da Computação) – Universidade de São Paulo, São Paulo, SP.

Souza, Cláudio Morgado de. **Visualg.** Disponível em http://www.apoioinformatica.inf.br/visualg/, acessado em 18/05/2009.

Sun Microsystems, Inc. **Java BluePrints: Model-View-Controller**. Disponível em http://java.sun.com/blueprints/patterns/MVC-detailed.html, acessado em 31/05/2009.

VanLehn, K., Lynch, C., Schulze, K., Shapiro, J.A., Shelby, R., Taylor, L., Treacy, D., Weinstein, A., and Wintersgill, M. (2005). **The Andes Physics Tutoring System: Lessons Learned**. *International Journal of Artificial Intelligence and Education*, 15 (3).

Vanlehn, K., Lynch, C., Schulze, K., Shapiro, J. A., Shelby, R. H., Taylor, L., Treacy, D. J., Weinstein, A., and Wintersgill, M. C. (2005). **The Andes physics tutoring system: Five years of evaluations.** In: G. I. McCalla and C.-K. Looi (Eds.), *Proceedings of the Artificial Intelligence in Education Conference*. Amsterdam: IOS.

Vanlehn, K. 2006. **The Behavior of Tutoring Systems. Int. Journal of Artificial Intelligence.** Ed. 16, 3 (Aug. 2006), 227-265.

Webber, C., Boff, E., Bono, F. **Ferramenta Especialista para Avaliação de Software Educacional**. In: Anais do Simpósio Brasileiro de Informática na Educação (SBIE 2009), Florianópolis, SC, 2009.

WEBportugol. Disponível em http://www.univali.br/webportugol, acessado em 18/05/2009.

Zenato, Matheus. **Proposta de Interface para um Ambiente de Aprendizagem de Programação de Computadores..** 2008. Trabalho de Conclusão de Curso. (Graduação em Bacharelado em Ciência da Computação) - Universidade de Caxias do Sul, Caxias do Sul, RS.