VIII Simpósio de Iniciação Científica 2025

"Do conhecimento acadêmico à transformação sustentável: inovação com validação científica"

Software de Diagramação com Geração Automática de Documentos de Classes

Bruna C. Alves[1] (EG), Lucas L. B. Barbosa[1] (EG), Lucas P. Souza[1] (EG), Egon L. Müller Júnior (PQ)[1]
[1]*Universidade Federal de Itajubá.*

Palavras-chave: UML. Desenvolvimento de Software. Automatização.

Introdução

O uso de diagramas de classes é um recurso essencial nos processos de desenvolvimento de software, possibilitando uma representação estrutural de sistemas e facilitando a comunicação entre equipes técnicas. Essa prática está alinhada ao que foi consolidado por Gamma et al. (1995), no clássico *Design Patterns: Elements of Reusable Object-Oriented Software*, que destaca a importância de padrões de projeto como soluções reutilizáveis e organizadas no desenvolvimento orientado a objetos. Contudo, observa-se que muitos desenvolvedores iniciantes, ou mesmo aqueles que não possuem familiaridade com práticas consolidadas da engenharia de software, encontram dificuldades em compreender a aplicabilidade e a importância dessa ferramenta.

No mercado já existem ferramentas com funcionalidades semelhantes ao sistema aqui proposto, como o *Umbrello UML Modeller*, software open source que permite a modelagem de diagramas e a geração de código, porém com uma interface pouco intuitiva e que exige maior conhecimento técnico para o uso eficiente. Outro exemplo, é o *Astah UML*, utilizado em ambientes acadêmicos, mas cuja versão completa é paga. Em comparação, a solução proposta neste trabalho busca oferecer uma alternativa gratuita, acessível e simplificada, com foco em estudantes e desenvolvedores em estágio inicial, unindo modelagem de diagramas de classe com geração automática de código em diferentes linguagens de forma prática e ágil.

Diante desse cenário, propôs-se o desenvolvimento de uma ferramenta que permita a geração do código base, que é representado pelos diagramas, automatizando o processo inicial da implementação de softwares.

O principal objetivo da pesquisa é criar um software capaz de entender a estrutura dos Diagramas de Classes UML, possibilitando a construção gráfica de diagramas de classes, e sendo capaz de gerar automaticamente os arquivos de código-fonte correspondentes em uma linguagem selecionada. Tal iniciativa justifica-se pela necessidade de tornar o processo de documentação e compreensão de sistemas mais ágil, didático e eficiente, tanto no ambiente acadêmico quanto no profissional.

Para atingir esses objetivos, adotou-se como método o planejamento colaborativo com levantamento de requisitos em reuniões de equipe, seguido pelo desenvolvimento iterativo baseado em metodologias ágeis. O processo contempla a implementação de uma interface gráfica para a geração automática de código a partir de uma descrição de um diagrama UML e, posteriormente, a disponibilização de documentação e materiais de treinamento – permitindo que outras pessoas compreendam e possam contribuir com o projeto futuramente. Dessa forma, espera-se oferecer uma solução que una praticidade e inovação, contribuindo para a formação de estudantes e a otimização do trabalho de desenvolvedores.

Metodologia

Para a execução do projeto, foram definidas 3 (três) etapas principais: 1) a análise da proposta e levantamento de requisitos para o desenvolvimento da aplicação; 2) a implementação do projeto; 3) a criação de materiais que documentam o projeto para instruir o uso do mesmo e para auxiliar pessoas que tenham o interesse em desenvolver um sistema semelhante ou contribuir com essa aplicação.

Na primeira etapa, durante as primeiras reuniões da equipe, foi feito um alinhamento de ideias e capacidade técnica entre os membros, para assim definir-se quais tecnologias seriam empregadas no desenvolvimento do projeto. Como resultado, foram definidos dois requisitos de alta importância: a análise de arquivos descritivos dos diagramas e a automatização da implementação dos diagramas. Assim, foi adotada a linguagem de programação Python para realizar a construção desse sistema. Essa escolha foi dada pela facilidade que o Python tem para gerenciar arquivos, por sua fácil modularização de códigos e por contar com a biblioteca *Python Lex Yacc (PLY)* que permite implementar analisadores léxicos, que transformam as descrições dos diagramas em tokens, e analisadores sintáticos que impõe regras gramaticais aos tokens.

Para padronizar uma gramática que descreve-se os

diagramas no formato UML, foi adotado a linguagem PlantUML como padrão, por ser uma ferramenta open source que faz a geração de diagramas UML baseado em uma descrição textual intuitiva – normalmente os arquivos de diagramas UML são descritos em formato *JSON* (*JavaScript Object Notation*) ou *XML* (*Extensible Markup Language*), dificultando a leitura por pessoas, porém facilitando o processamento por máquina. Além de facilitar a compreensão dos arquivos descritivos, o PlantUML já faz o desenho dos diagramas, o que sana um requisito secundário do sistema: a visualização dos diagramas.

Tendo como definida as ferramentas utilizadas pelo grupo e os requisitos necessários para o desenvolvimento da aplicação, foi iniciada a segunda etapa do projeto. Durante a implementação do sistema, primeiramente buscou-se compreender a gramática da descrição que é usada pelo PlantUML na geração dos diagramas de classes UML, para que então fosse possível implementar os analisadores léxico (chamado de *lexer*) e sintático (chamado de *parser*), seguindo os princípios clássicos de construção de compiladores (AHO; SETHI; ULLMAN, 1995; LOUDEN, 2004).

Em seguida foram implementados três geradores de código: um para Python, outro para C# e um para Java, que podem ser intercalados utilizando um Interface em Linha de Comando (mais conhecido como *CLI* do inglês). Os geradores recebem os objetos que, após a análise explicada anteriormente, representam os diagramas e seguem cada um a regra de uma linguagem para realizar a conversão das informações do diagrama para um código escrito na linguagem definida. Após a conversão os códigos são salvos em arquivos que também são dispostos em diretórios seguindo a estrutura que consta nos diagramas.O fluxo completo deste processo é ilustrado na Figura 1.

Após isso, foi implementada uma interface gráfica, que torna mais fácil e simples o uso dessa aplicação por pessoas leigas em programação. A interface foi implementada utilizando HTML, CSS e JavaScript, além de utilizar a biblioteca *PyWebView* que faz o empacotamento e cuida da renderização da interface descrita pelos arquivos. E, por fim, após refatorações no código fonte do sistema, o grupo gerou uma documentação que detalha a implementação do mesmo, como o desenvolvimento do analisador léxico e dos módulos geradores de código.

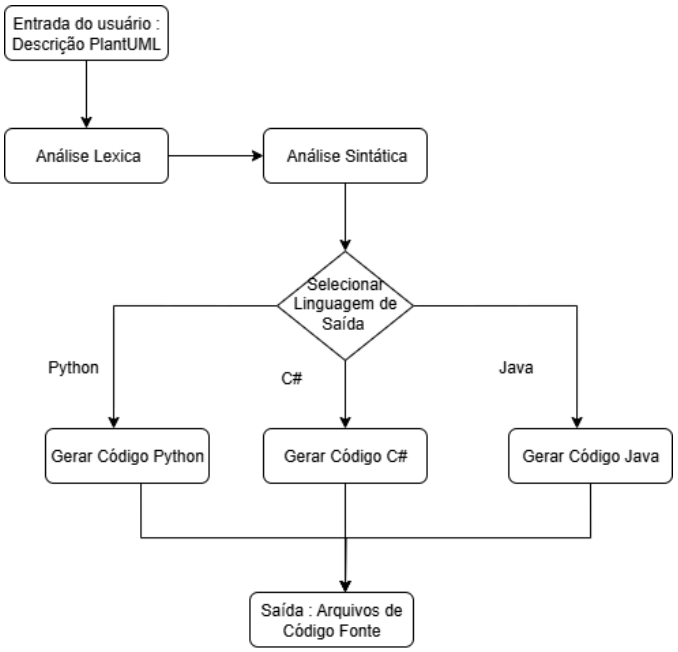

Figura 1 – Fluxograma do processo de análise e geração de código.

Resultados e discussão

Ao passar pela análise sintática, os elementos do diagrama são convertidos para classes. Cada um desses elementos possui uma classe específica que é utilizada para que os geradores de código diferenciem as informações passadas, como por exemplo saber que um elemento se trata do nome de uma classe por ser do tipo *plantuml_classe* ou que se trata de um atributo por ser do tipo *plantuml_atributo*. É importante ressaltar que, devido a abordagem adotada neste processamento da descrição dos diagramas, o software funciona, basicamente, como um compilador, sendo assim necessário que os diagramas sejam escritos seguindo a estrutura do PlantUML para descrição de diagramas de classes.

Na interface de usuário, exibida na Figura 2, basta informar a descrição do diagrama no canto esquerdo, selecionar uma linguagem (por padrão é deixado no Python) e então apertar o botão para realizar a conversão. A listagem dos arquivos convertidos será exibida a direita. Caso deseje visualizar um arquivo basta clicar sobre o nome do arquivo e o código implementado será exibido no lugar da listagem dos arquivos.

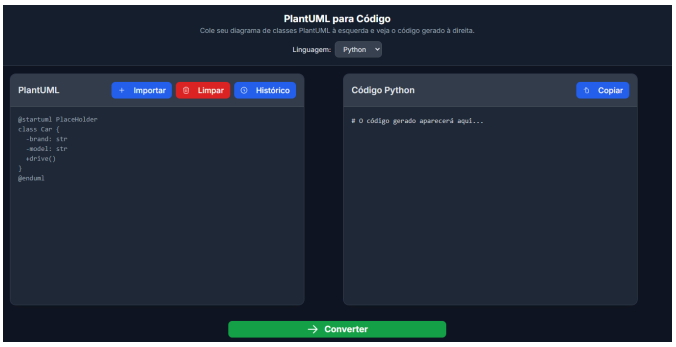

Figura 2 – Interface gráfica da aplicação

Para validar a eficácia da ferramenta, foram realizados testes com diferentes modelos de diagramas de classe. Como exemplo ilustrativo, a Figura 3 demonstra a conversão de um diagrama de classe simples, que modela o objeto carro. A descrição em PlantUML fornecida à ferramenta (à esquerda) foi corretamente interpretada pelos analisadores léxico e sintático, gerando o código C# funcional (à direita), que respeita os atributos, métodos e seus respectivos níveis de visibilidade (público, privado e protegido).

Figura 3 - Exemplo classe carro em C#

Esse resultado confirma que o sistema foi capaz de interpretar corretamente atributos e métodos públicos, protegidos e privados descritos no PlantUML, convertendo-os em código Python válido e funcional.

A validação da ferramenta foi realizada por meio de uma suíte de testes unitários automatizados, que cobriram os componentes críticos do back-end. Foram criados casos de teste específicos para avaliar: O parser, garantindo a correta interpretação da sintaxe PlantUML e a extração de todos os elementos do diagrama; o gerador de código, confirmando a tradução fiel do modelo intermediário para Python, C# e Java; a conversão de estruturas fundamentais da Orientação a Objetos, como interfaces, enuns (enumerações) e diferentes tipos de relacionamentos (herança, associação, etc.); a correta organização do código-fonte gerado em pacotes (ou *namespaces*), conforme especificado no diagrama.

Para cada cenário, foram utilizados arquivos de entrada do diretório plantUML_for_tests, permitindo comparar a saída gerada com o resultado esperado. A execução bem-sucedida desta suíte de testes assegurou a robustez e a precisão do processo de conversão da ferramenta.

A principal restrição da ferramenta é sua especialização em Diagramas de Classe UML. A conversão é otimizada para interpretar a estrutura de classes, atributos, métodos, pacotes, enums, interfaces e os principais tipos de relacionamentos (como herança e associação), conforme validado pela suíte de testes do projeto. Adicionalmente, os testes de desempenho indicaram uma alta eficiência, com o tempo médio de execução para a análise e geração de código sendo de aproximadamente 0,22 segundos. Embora o número de linguagens suportadas ainda seja limitado, essa notável agilidade posiciona a ferramenta como uma ótima opção para acelerar as fases iniciais de desenvolvimento de software.

Atualmente, o sistema não oferece suporte para a conversão de outros tipos de diagramas UML, como Diagramas de Sequência ou de Casos de Uso, sendo esta uma clara oportunidade de expansão futura.

Conclusões

Dessa forma, a ferramenta contribui para a consolidação de práticas sólidas de modelagem e desenvolvimento, em consonância com os princípios apresentados por Gamma et al. (1995), que destacam a relevância da orientação a objetos e do uso de padrões de projeto como elementos fundamentais para a construção de software robusto e reutilizável.

O trabalho resultou no desenvolvimento bem-sucedido de uma aplicação capaz de automatizar a geração de código-fonte a partir de descrições textuais de Diagramas de Classe UML, utilizando a sintaxe PlantUML. Os resultados obtidos confirmam a viabilidade da abordagem proposta, demonstrando que a utilização de analisadores léxicos e sintáticos customizados constitui uma solução eficaz para a interpretação e tradução de modelos de software.

O principal êxito do projeto foi estabelecer uma ponte eficiente entre o design de alto nível e a codificação inicial, reduzindo o esforço manual e repetitivo, minimizando a ocorrência de erros de transcrição e incentivando a adoção de boas práticas de engenharia de

software desde as etapas iniciais do desenvolvimento. A interface gráfica desenvolvida amplia o acesso à ferramenta, tornando seu uso viável tanto para estudantes em processo de aprendizagem sobre UML e orientação a objetos quanto para desenvolvedores que desejam agilizar seus fluxos de trabalho.

Como perspectivas futuras, vislumbra-se a expansão da ferramenta para contemplar outros tipos de diagramas UML, como Diagramas de Sequência e de Casos de Uso. Além disso, propõe-se a inclusão de novas linguagens de programação para a geração de código, como TypeScript e C++, bem como a implementação de um editor de texto integrado com recursos de realce de sintaxe e autocompletar, otimizando a experiência na escrita das descrições em PlantUML.

Agradecimentos

Agradecemos à Universidade Federal de Itajubá (UNIFEI), ao Programa de Educação Tutorial (PET) e ao Fundo Nacional de Desenvolvimento da Educação (FNDE) pelo apoio institucional e financeiro que possibilitaram a realização deste projeto. Estendemos nossos agradecimentos ao grupo PET-TEC e ao coordenador Prof. Egon Luiz Muller Junior pela orientação, incentivo e dedicação ao longo de todas as etapas de desenvolvimento. O suporte e a oportunidade oferecidos foram fundamentais para a concretização deste trabalho.

Referências

Class Diagram syntax and features. Disponível em: <https://plantuml.com/class-diagram>. Acesso em: 22 ago. 2025.

Introduction | pywebview. Disponível em: <https://pywebview.flowrl.com/guide/>. Acesso em: 22 ago. 2025.

Welcome to Python.org. Disponível em: <https://www.python.org/>. Acesso em: 22 ago. 2025.

GAMMA, Erich; HELM, Richard; JOHNSON, Ralph; VLISSIDES, John. **Design Patterns: Elements of Reusable Object-Oriented Software. Reading**, MA: Addison-Wesley, 1995.

AHO, Alfred V.; SETHI, Ravi; ULLMAN, Jeffrey D. ***Compiladores: Princípios, Técnicas e Ferramentas.*** Rio de Janeiro: Livros Técnicos e Científicos, 1995.

LOUDEN, Kenneth C. ***Compiladores: Princípios e Práticas.*** Porto Alegre: Pioneira Thomson Learning, 2004.