DSL Máscara Lang

1st Arthur Bortoloti Ruiz
Curso de engenharia de computação
Fundação Herminio Hometto
Araras, Brasil
arthur.ruiz@alunos.fho.edu.br

1st Caio Henrique Yabuki Souza Rio
Curso de engenharia de computação
Fundação Herminio Hometto
Araras, Brasil
caioyabuki@alunos.fho.edu.br

1st Matteo Carbinatto
Curso de engenharia de computação
Fundação Herminio Hometto
Araras, Brasil
matteocarbinatto@alunos.fho.edu.br

I. Análise do Artigo-Base

O artigo *Implementation of Structured Object-Oriented Formal Language for Warehouse Management System* aborda como problema central as dificuldades encontradas na especificação e desenvolvimento de sistemas críticos, como os sistemas de gerenciamento de armazéns (WMS). Tais sistemas estão sujeitos a erros provenientes de especificações ambíguas, incompletas ou inconsistentes, que afetam diretamente a implementação e a manutenção do software. Como solução, os autores propõem o uso da linguagem formal SOFL (Structured Object-Oriented Formal Language), que visa garantir maior clareza, precisão e manutenibilidade durante o processo de desenvolvimento.

A metodologia apresentada baseia-se na aplicação da abordagem SOFL, estruturada em três níveis de abstração:

- **Especificação informal**: definição dos requisitos em linguagem natural;
- **Especificação semi-formal**: representação intermediária usando palavras-chave formais;
- **Especificação formal**: descrição rigorosa da arquitetura do sistema, baseada em lógica matemática, utilizando diagramas CDFD (Condition Data Flow Diagram) e pré/pós-condições.

Os autores aplicam essa metodologia ao desenvolvimento de um sistema de gerenciamento de armazéns e validam a proposta por meio de métricas como a complexidade ciclomática, índice de manutenibilidade (MI) e análise por ponto de função (Function Point). Os resultados demonstram um aumento de 56,94% no índice de manutenibilidade e uma redução significativa na complexidade do código, evidenciando os benefícios da aplicação de técnicas formais no desenvolvimento de software seguro e robusto.

Diversos conceitos do artigo podem ser aproveitados no projeto *MascaraLang*, como:

- A abordagem em camadas (informal $\rightarrow$ formal), que inspirou a separação entre léxico, parser e execução;
- A preocupação com clareza semântica e rastreabilidade, refletida na estruturação das regras e detecção de erros;
- A importância da modularização, que motivou a organização do código para futura expansão da linguagem.

Dessa forma, o artigo-base serviu como referência conceitual sólida para o desenvolvimento da linguagem de domínio específico proposta, orientando tanto a arquitetura quanto os princípios de qualidade e robustez do sistema.

II. Definição do Novo Contexto

O projeto *MascaraLang* propõe o desenvolvimento de uma linguagem de domínio específico (DSL – Domain-Specific Language) voltada à simulação de partidas de um jogo chamado *Mascarade*, um boardgame de Bruno Faidutti, jogo de personagens ocultos com ações baseadas em influência e estratégia. A linguagem permite descrever, de forma textual e estruturada, o início de uma partida com a criação de jogadores e a sequência de ações possíveis, que são analisadas sintática e semanticamente e, em seguida, executadas automaticamente pelo interpretador da linguagem.

A. Delimitação do Problema

O objetivo principal deste trabalho é criar uma linguagem que permita representar e simular partidas de forma automatizada, clara e segura. O escopo delimitado abrange:

- A criação de jogadores e verificação de suas declarações;
- A análise léxica e sintática de comandos que descrevem as ações do jogo;
- A execução interpretativa das ações com base nas regras do jogo;
- A simulação de um fluxo de partida completo, com suporte a ações como *habilidade*, *troca*, *revelar* e *contestar*.

A implementação foi realizada utilizando Python para o interpretador e análise léxica, com entrada em arquivos no formato `.cpp`, contendo os comandos da DSL.

B. Relevância do Contexto

O uso de linguagens específicas de domínio em jogos e simulações tem crescido significativamente por sua capacidade de simplificar a representação de regras complexas e permitir automação de testes e simulações. O *MascaraLang* oferece uma abordagem declarativa para o controle de partidas, permitindo:

- Redução de ambiguidades na execução das regras do jogo;
- Facilidade de replicação de partidas para análise estratégica;
- Potencial aplicação em inteligência artificial para treinamento e tomada de decisão;

- Adoção em contextos educacionais como ferramenta de apoio ao ensino de lógica, linguagens formais e compiladores.

C. Conexão com o Artigo-Base

A escolha da abordagem de desenvolvimento do *MascaraLang* foi fortemente influenciada pelo artigo-base, que demonstrou os benefícios da aplicação de métodos formais no desenvolvimento de sistemas críticos. Tal como o artigo utilizou a metodologia SOFL com divisão em especificações informal, semi-formal e formal, o *MascaraLang* também estrutura sua implementação de forma modular:

- O *lexer.py* realiza a análise léxica com base em expressões regulares (nível informal);
- O *parser.py* executa a análise sintática e parte da semântica (nível semi-formal);
- A execução das ações é feita por meio de regras semânticas rigorosas, inspiradas em pré e pós-condições (nível formal).

Dessa forma, a metodologia do artigo foi reinterpretada e adaptada ao contexto de jogos com personagens ocultos, provando-se útil para garantir clareza estrutural, rastreabilidade de ações e robustez na execução da linguagem desenvolvida.

III. Adaptação da Abordagem Metodológica

A metodologia utilizada no artigo-base se apoia no uso da linguagem formal SOFL (Structured Object-Oriented Formal Language), composta por três níveis de especificação: informal, semi-formal e formal. A aplicação foi voltada ao desenvolvimento de um sistema crítico (Warehouse Management System – WMS), envolvendo controle rígido de dados e uso de técnicas matemáticas para garantir a correção das operações.

No contexto do *MascaraLang*, algumas adaptações metodológicas foram necessárias para atender à natureza diferente do domínio – um jogo com regras lógicas, ações sequenciais e múltiplos agentes interativos. A seguir, são listadas as principais modificações realizadas em relação à metodologia original:

A. Modificações Realizadas

- **Substituição da linguagem formal SOFL por uma DSL textual interpretada:** ao invés de utilizar notações matemáticas e estruturas de formalização como CDFD, foi criada uma gramática personalizada, utilizando análise léxica e sintática com Python e PLY.
- **Execução interpretativa direta ao invés de geração de código:** enquanto o artigo original propõe uma formalização que pode ser posteriormente transformada em código executável, o *MascaraLang* adota um modelo interpretado, em que os comandos são lidos e executados diretamente.
- **Foco na simulação de domínio lúdico em vez de sistema crítico:** ao invés de priorizar controle de falhas ou segurança (como em sistemas WMS), o foco está na fidelidade à lógica do jogo e na clareza das regras e interações.
- **Redução de camadas formais:** apesar da inspiração na estrutura em camadas da SOFL, a implementação foi simplificada para duas camadas principais: léxica/sintática (via parser) e semântica/execução (via regras no interpretador).
- **Eliminação de métricas de software (MI, CC):** diferentemente do artigo original, que utiliza métricas como complexidade ciclomática e índice de manutenibilidade, o projeto não requer tais medições formais, dado que seu objetivo principal é comportamental (simulação de partidas).

B. Justificativas das Modificações

As alterações metodológicas foram necessárias por conta das seguintes características do novo contexto:

- **Domínio diferente:** o contexto de jogos de turno com múltiplos jogadores não exige a mesma rigidez de verificação formal de um sistema crítico; requer, no entanto, clareza, simulação precisa e flexibilidade.
- **Foco em execução e feedback imediato:** a escolha por uma linguagem interpretada permite que os comandos sejam executados diretamente, com saída imediata ao usuário, o que é essencial em contextos lúdicos e interativos.
- **Viabilidade e escopo acadêmico:** a adoção de um modelo leve, baseado em Python e análise sintática customizada, permite o desenvolvimento dentro do escopo do projeto, sem a complexidade de frameworks formais.
- **Prioridade em usabilidade e extensibilidade:** a linguagem textual e as ações codificadas foram projetadas para facilitar futuras adições, como novos personagens ou comandos, algo mais complexo no modelo formal da SOFL.

Essas adaptações preservam a essência da proposta do artigo-base — modularidade, clareza estrutural e separação de responsabilidades — enquanto permitem uma implementação prática e adequada ao novo domínio de aplicação.

IV. Justificativas das Escolhas Técnicas e Avaliação

O desenvolvimento da aplicação *MascaraLang* foi realizado com base em uma abordagem iterativa e incremental, seguindo os princípios de construção modular inspirados na metodologia SOFL, adaptada ao contexto de uma linguagem de domínio específico (DSL). O objetivo do desenvolvimento foi permitir a leitura, análise e execução automática de comandos que simulam uma partida de jogo de influência e personagens ocultos.

A. Etapas do Desenvolvimento

O processo de implementação foi dividido nas seguintes etapas:

1) **Definição da Gramática da Linguagem**
   Foram especificados os tokens principais da DSL (*JOGADOR, ACAO, PERSONAGEM, NOVA_INSTANCIA, etc.*) utilizando expressões regulares no módulo

`lexer.py`. Essa etapa definiu a estrutura básica da linguagem a ser interpretada.

2) **Implementação do Analisador Léxico**
   A biblioteca **PLY (Python Lex-Yacc)** foi utilizada para construir o analisador léxico, que lê os comandos do arquivo fonte (com extensão `.cpp`) e transforma o conteúdo em uma sequência de tokens. Essa leitura é feita no arquivo `lexer.py` pela função `inicializaLexer()`.
3) **Desenvolvimento do Analisador Sintático e Semântico**
   O arquivo `parser.py` é responsável por realizar a análise sintática e semântica dos tokens produzidos pelo lexer. Cada comando é verificado quanto à conformidade gramatical e executado por meio de funções específicas. A semântica da linguagem é definida com base em ações do jogo, como *habilidade, troca, contestar* e *revelar*, cujas regras estão programadas diretamente nas funções Python.
4) **Execução Interpretada dos Comandos**
   Após a análise, os comandos são executados em tempo real, atualizando o estado do jogo (como o número de moedas, personagens em jogo e resultado de ações). As funções simulam o comportamento do jogo de forma automatizada e seguem a lógica dos personagens e regras previstas.
5) **Testes com Roteiros de Partida**
   Foram criados arquivos de teste contendo scripts de partidas completas, como o `main.cpp`, que contém a instância dos jogadores e as ações executadas por eles. Esses scripts foram utilizados para validar a execução correta da linguagem e detectar possíveis erros sintáticos e semânticos.
6) **Prototipação e Validação**
   Com o interpretador em funcionamento, foram realizadas simulações completas com diferentes sequências de ações, verificando:
   - Declarações inválidas de jogadores;
   - Ações não permitidas ou malformadas;
   - Condições de vitória e manipulação de moedas;
   - Fluxo completo de uma partida.
7) **Planejamento de Extensões**
   O projeto prevê futuras melhorias, como modularização das ações, adição de novos personagens, backend translational (geração de código) e integração com editores para destaque de sintaxe.

B. Ferramentas e Tecnologias Utilizadas

- **Linguagem de Programação:** Python 3.11
- **Biblioteca de análise léxica: PLY (Python Lex-Yacc)** para construção do lexer
- **Editor de texto:** VS Code, com extensões para Python
- **Sistema operacional:** Compatível com Windows e Linux
- **Scripts de entrada:** Arquivos `.cpp` contendo os comandos da DSL

O desenvolvimento seguiu uma abordagem iterativa, com protótipos sendo ajustados progressivamente conforme os testes revelavam novas necessidades ou inconsistências. A clareza da gramática e a separação entre análise e execução permitiram organizar o projeto de maneira escalável e reutilizável.

V. Justificativas das Escolhas Técnicas e Avaliação

O desenvolvimento da *MascaraLang* envolveu uma série de decisões técnicas que foram fundamentadas nas necessidades do projeto, nos princípios da metodologia base e nas características do domínio escolhido. As escolhas buscaram equilibrar clareza, simplicidade de implementação e possibilidade de extensão futura.

A. Justificativa das Tecnologias Utilizadas

- **Linguagem Python:** escolhida por sua sintaxe acessível, ampla adoção acadêmica e facilidade de manipulação de arquivos, estruturas de dados e integração com bibliotecas de análise léxica e sintática.
- **PLY (Python Lex-Yacc):** biblioteca leve e eficaz para construção de analisadores léxicos e sintáticos. Inspirada no modelo tradicional do YACC, permite definir a gramática da linguagem com expressões regulares e regras de produção, sem dependências externas complexas.
- **Arquivos `.cpp` como entrada:** escolha deliberada para simular um ambiente de programação real, onde comandos são descritos em uma linguagem textual clara e de fácil edição.
- **Estrutura interpretativa:** em vez de compilar ou gerar código, a DSL executa os comandos diretamente após a análise sintática, o que acelera o ciclo de desenvolvimento, teste e validação de partidas.
- **Separação de responsabilidades:** o projeto foi dividido em três módulos principais – `lexer.py`, `parser.py` e o script de entrada – inspirando-se na abordagem modular da SOFL e permitindo melhor organização e manutenção do código.

B. Critérios de Avaliação da Aplicação

Para considerar a aplicação funcional e bem-sucedida, foram adotados os seguintes critérios:

- **Correção sintática e semântica:** o sistema é capaz de detectar comandos malformados ou inválidos, identificando erros de sintaxe ou uso incorreto de jogadores e personagens.
- **Execução de fluxo completo:** a linguagem permite simular uma partida completa, com ações variadas, manipulação de estado (moedas, personagens) e condições de vitória.
- **Feedback ao usuário:** todas as ações geram saídas descritivas no console, permitindo acompanhar o andamento da partida e entender os efeitos das ações.
- **Manutenção de estado:** o interpretador mantém uma memória consistente dos jogadores e suas variáveis

(posição, personagem, moedas), simulando corretamente o contexto de um jogo de cartas com múltiplos agentes.
- **Robustez de regras:** as ações são validadas com base nas regras semânticas do jogo. Por exemplo, o personagem *benfeitor* afeta jogadores adjacentes; o *traidor* vence com 10 moedas; a *viúva* só recebe moedas se tiver menos de 10, etc.
- **Reprodutibilidade:** a aplicação pode ser executada múltiplas vezes com diferentes entradas, mantendo a lógica e comportamento previsíveis.

C. Estado Final Apresentado

O estado apresentado pela aplicação ao final da execução consiste em:

- Exibição dos jogadores com seus respectivos atributos (identificador, personagem, quantidade de moedas, posição);
- Histórico impresso de ações realizadas e seus efeitos;
- Indicação de erro sintático ou semântico, caso ocorra;
- Declaração do jogador vencedor, se alguma condição de vitória for atingida.

Esses elementos demonstram que a DSL desenvolvida cumpre seu propósito como ferramenta de simulação e representação formal de partidas, validando a aplicação prática da metodologia analisada e das decisões técnicas implementadas.

REFERENCES

[1] I. Afifudin and I. Martina, "Implementation of Structured Object-Oriented Formal Language for Warehouse Management System," *CommIT (Communication and Information Technology) Journal*, vol. 14, no. 1, pp. 1–8, 2020.

[2] A. Negm, M. Elbendary, and A. E. Hassanien, "Survey on Domain-Specific Languages — Implementation Aspects," *Proceedings of the 14th International Conference on Computer Engineering and Systems (ICCES)*, Cairo, Egypt, pp. 465–470, Dec. 2019.

[3] M. Fowler, "A Pedagogical Framework for Domain-Specific Languages," *IEEE Software*, vol. 26, no. 4, pp. 13–15, Jul./Aug. 2009. [Online].

[4] H. Prähofer, D. Hurnaus, C. Wirth, and H. Mössenböck, "The Domain-Specific Language Monaco and its Visual Interactive Programming Environment," in *Proc. IEEE Symposium on Visual Languages and Human-Centric Computing (VLHCC)*, 2007, pp. 104–107.

[5] J. L. Sierra, B. Fernández-Manjón, A. Fernández-Valmayor, and A. Navarro, "Document-oriented Software Construction based on Domain-Specific Markup Languages," in *Proc. Int. Conf. on Information Technology: Coding and Computing (ITCC)*, 2005, pp. 514–519.

[6] A. B. Ruiz, "DSL-Mascara-Lang", GitHub repository, 2025. [Online]. Available: https://github.com/ABRuizz/DSL-Mascara-Lang.git