Validação de programação para sistemas de irrigação com Domain Specific Languages

1st Luan Felipe Abiles Pinto
Fundação Hermínio Ometto
RA:109902
luanfap2000@alunos.fho.edu.br

2nd Matheus Carlos Andrade Setin
Fundação Hermínio Ometto
RA:109606
setin@alunos.fho.edu.br

3rd Willian Henrique de Lima
Fundação Hermínio Ometto
RA:109408
willianhl@alunos.fho.edu.br

1 Introdução

A crescente demanda por soluções inteligentes no meio rural tem impulsionado o desenvolvimento de sistemas de automação voltados à agricultura de precisão. Entre essas soluções, os sistemas de irrigação automatizados se destacam por sua capacidade de otimizar o uso dos recursos hídricos, aumentar a produtividade e minimizar desperdícios. No entanto, a programação desses sistemas ainda representa um obstáculo para muitos usuários que não possuem familiaridade com linguagens de programação tradicionais (SILVA, 2010).

Diante desse cenário, este projeto propõe o desenvolvimento de uma Linguagem Específica de Domínio (DSL – Domain-Specific Language) voltada para o controle de sistemas de irrigação automatizados. A linguagem foi concebida com uma sintaxe simples e acessível, permitindo que operadores agrícolas, técnicos ou produtores rurais definam, de forma intuitiva, condições de operação, leitura de sensores e comandos temporizados, sem a necessidade de conhecimentos técnicos aprofundados (SPINELLIS, 2001).

A execução dos códigos definidos na DSL será realizada por uma máquina virtual interpretativa. Além disso, o sistema contará com uma arquitetura de comunicação baseada em MQTT, integrando o computador, os dispositivos de campo e uma Interface Homem-Máquina (IHM), o que amplia a aplicabilidade e a flexibilidade do projeto em ambientes reais, inclusive em regiões com infraestrutura tecnológica limitada (GROENEVELD, 2021).

2 Objetivo

O principal objetivo deste projeto é desenvolver um sistema de irrigação automatizado acessível a usuários sem conhecimentos avançados em programação, por meio da criação de uma Linguagem Específica de Domínio (DSL). Essa linguagem foi concebida para permitir que operadores agrícolas, técnicos ou produtores definam, de forma simples e intuitiva, rotinas de irrigação baseadas em condições ambientais, leituras de sensores e temporizações, viabilizando o controle eficiente dos recursos hídricos e a automação inteligente no campo (TESTEZLAF, 2017).

Além disso, busca-se garantir que a linguagem desenvolvida seja compatível com sistemas embarcados de baixo custo e baixo consumo energético, como por exemplo, o Raspberry Pi. Para isso, será implementada uma máquina virtual interpretativa capaz de processar os comandos da DSL diretamente no hardware embarcado. Essa abordagem visa oferecer uma solução prática, portátil e adequada a diversos cenários agrícolas, inclusive em regiões com infraestrutura tecnológica limitada.

Complementando a proposta, o projeto estabelece um fluxo completo de interpretação, com etapas de análise léxica, sintática e semântica do código, assegurando a validação correta das instruções fornecidas pelo usuário. O sistema fornecerá mensagens de erro claras e orientativas, facilitando a identificação e correção de falhas pelo próprio operador, mesmo sem conhecimento técnico aprofundado.

Com o intuito de ampliar a aplicabilidade e a conectividade do sistema, o projeto também integra uma arquitetura de comunicação baseada no protocolo MQTT. Essa arquitetura conecta o computador que armazena e envia os comandos da DSL a uma Interface Homem-Máquina (IHM) por meio de um servidor Broker Esp-32, permitindo que sensores e atuadores sejam controlados e monitorados em tempo real. O computador atua como publicador de comandos e assinante dos dados de sensores, enquanto a IHM assume o papel de publicadora dos sensores e assinante das atualizações de controle, viabilizando a configuração remota e amigável dos parâmetros do sistema.

Por fim, a solução será validada por meio de testes em ambientes simulados e em bancadas físicas com sensores reais de umidade, temperatura e bombas. A avaliação prática visa comprovar a funcionalidade da linguagem, a precisão na interpretação dos comandos e a confiabilidade da comunicação em condições próximas às encontradas no campo. Espera-se, assim, entregar uma ferramenta funcional, compreensível e de fácil adoção, que contribua de forma efetiva para a modernização da agricultura por meio da tecnologia embarcada e conectada.

3 Metodologia

O desenvolvimento deste projeto seguiu uma abordagem estruturada, baseada nos princípios de construção de linguagens formais, comunicação em rede e implementação de sistemas embarcados. Inicialmente, foi concebida uma linguagem específica de domínio (DSL – Domain-Specific Language), voltada para automação de irrigação agrícola. Essa

linguagem textual foi desenhada com uma sintaxe simples e de fácil leitura, permitindo a operadores sem conhecimentos em programação tradicional descreverem, de forma clara, rotinas como aguardar determinados intervalos, realizar leituras condicionais de sensores e acionar atuadores. Comandos como ligar bombas e leitura de sensores foram projetados para serem de fácil implementação e entendimento (ROUHANI, 2015).

Após a definição da linguagem, iniciou-se a implementação do compilador, cuja estrutura foi dividida em três fases principais: análise léxica, sintática e semântica. O analisador léxico tem a função de identificar os componentes básicos do código-fonte, como palavras-chave, operadores, números e identificadores, gerando uma sequência de tokens que representam essas unidades. Em seguida, o analisador sintático verifica se esses tokens estão organizados segundo as regras gramaticais da linguagem, construindo uma estrutura hierárquica intermediária que representa a lógica do programa. Essa estrutura é então submetida ao analisador semântico, que avalia a coerência lógica das instruções, identificando inconsistências como comandos mal utilizados, variáveis não declaradas ou comparações inválidas. Essa fase também é responsável por fornecer mensagens de erro detalhadas e compreensíveis ao usuário.

Com o código validado, é realizada a geração de uma representação intermediária, que pode ser uma lista de instruções ou um conjunto de operações similares. Essa representação é processada por uma máquina virtual interpretativa desenvolvida especialmente para esse projeto. A máquina virtual é composta por um interpretador que reconhece comandos como ativação de bombas, espera por tempo, leitura de sensores e saltos condicionais.

O código interpretado pode ser armazenado em memória e executado localmente, o que possibilita o funcionamento do sistema sem a necessidade de um computador intermediário. Isso torna o projeto viável para aplicações em campo, com atuação direta em sistemas de irrigação reais, leitura de sensores como o DHT22 ou sensores de umidade do solo, e acionamento de LEDs.

Uma evolução significativa do projeto incluiu a integração de comunicação em rede utilizando o protocolo MQTT. O código-fonte é mantido no computador, que atua como cliente tanto na publicação de comandos (Publisher) quanto na inscrição para recebimento de dados dos sensores (Subscriber). A comunicação ocorre por meio de um servidor Broker esp-32, que funciona como o elo de integração entre o computador e a Interface Homem-Máquina (IHM). Essa IHM representa a interface gráfica pela qual o usuário pode configurar, de forma visual e intuitiva, os parâmetros de operação dos sensores de campo e dos atuadores como bombas e válvulas. A dinâmica de funcionamento da arquitetura implementada permite que, do computador para o Broker MQTT, haja a publicação dos comandos definidos na linguagem DSL, enquanto simultaneamente o computador se inscreve nos tópicos relacionados aos sensores para receber as informações de volta. O ESP32, por sua vez, se inscreve nos tópicos de comando para executar os procedimentos requisitados e envia os dados de sensores para o Broker, possibilitando sua publicação. A IHM, integrada ao sistema, publica as leituras dos sensores e também se inscreve para receber atualizações dos comandos, servindo como elo entre a percepção de campo e a configuração das ações. Essa abordagem garante comunicação bidirecional eficiente entre todos os elementos do sistema.

Durante a fase de análise semântica, diversos aspectos do programa de irrigação serão avaliados para garantir a consistência lógica e a segurança do sistema. Um dos principais pontos a serem verificados será a declaração dos sensores, assegurando que todos os sensores utilizados tenham sido previamente declarados, que os identificadores (IDs) sejam únicos e que os nomes atribuídos estejam dentro dos padrões esperados. Essa etapa será essencial para evitar erros de referência que comprometam a execução correta do programa. Além disso, serão analisadas as regras de irrigação, com foco em condições lógicas que determinarão a ativação ou desativação da bomba de água. A análise verificará se há conflitos entre as regras estabelecidas, como condições sobrepostas que possam gerar acionamentos contraditórios, e se os limites numéricos definidos estarão dentro de faixas válidas, como por exemplo, valores de umidade entre 0 % e 100 %. Dessa forma, buscar-se-á assegurar que as decisões tomadas pelo sistema estejam embasadas em condições coerentes e seguras. Outro aspecto importante será a validação dos tempos de espera, garantindo que os comandos esperar(tempo) estejam dentro de intervalos razoáveis. Também será verificado se a sequência das operações seguirá uma lógica compatível com a finalidade do sistema, prevenindo instruções que possam causar travamentos, atrasos excessivos ou comportamentos indesejados.

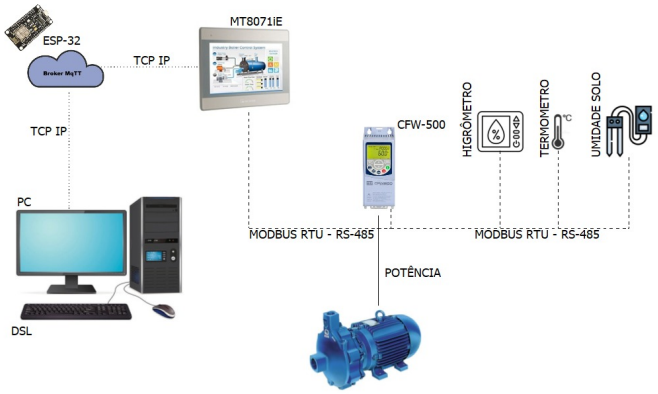

Fig. 1. Fluxograma para projeto funcionar com DSL

Para realizar essa verificação, serão aplicados testes semânticos específicos, por meio de uma função de execução chamada ”executar sistema irrigação”, que analisará o código antes de permitir sua execução. Os testes incluirão casos como programas semanticamente válidos, uso de sensores não declarados, repetição de IDs, regras conflitantes, limites inválidos de leitura e tempos de espera excessivos. Essa etapa será fundamental para garantir a robustez do sistema,

prevenindo falhas, protegendo os equipamentos e assegurando que as decisões automatizadas estejam alinhadas com as necessidades reais da irrigação agrícola. Por fim, o sistema será testado em ambiente simulado e também em bancada física com o computadores, sensores e o IHM, com o intuito de validar tanto a interpretação correta da linguagem quanto a resposta dos componentes físicos. Os testes incluirão cenários com comandos válidos e inválidos, medição de tempos de execução e simulação de diferentes condições de umidade, a fim de avaliar a eficácia da lógica condicional implementada na DSL. Esse processo de validação será fundamental para garantir que a linguagem proposta seja funcional, compreensível e aplicável em contextos reais de automação agrícola.

3.1 Funcionamento

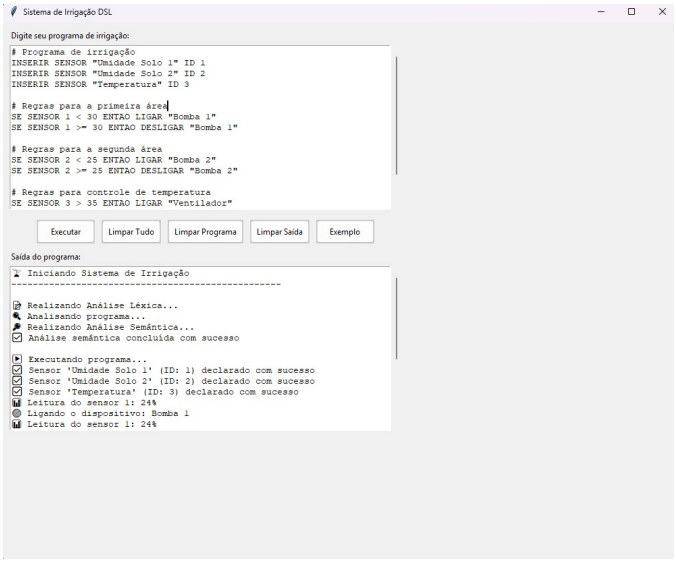

Fig. 2. Interface gráfica do sistema de irrigação baseado em DSL

O sistema de irrigação implementado opera através de um interpretador que processa comandos em português para controlar dispositivos de irrigação. O funcionamento ocorre em etapas sequenciais, onde cada fase é responsável por uma parte específica do processamento. Inicialmente, o Analisador Léxico recebe o texto do programa e realiza a tokenização, identificando cada elemento do código. Este processo é fundamental pois converte o texto em unidades significativas, como comandos (INSERIR, SE, ENTAO), identificadores de sensores, operadores de comparação e valores numéricos. Por exemplo, quando o sistema encontra a palavra ”SE”, ele a reconhece como um token do tipo PALAVRA IF, que será utilizado posteriormente para construir a lógica do programa.

Em seguida, o Analisador Sintático recebe estes tokens e constrói uma Árvore Sintática Abstrata (AST). Este processo verifica se a sequência de tokens forma uma estrutura válida, garantindo que os comandos estejam na ordem correta. Por exemplo, após um ”SE” deve vir um ”SENSOR”, seguido por um número de identificação, um operador de comparação e um valor limite.

O Analisador Semântico então realiza verificações mais profundas na estrutura criada. Ele confirma se todos os sensores referenciados foram previamente declarados, se as regras não se contradizem, e se os valores utilizados estão dentro de limites aceitáveis. Esta etapa é crucial para garantir a integridade e segurança do sistema.

Por fim, a Máquina Virtual executa o programa interpretando a AST. Durante a execução, ela simula as leituras dos sensores, aplica as regras definidas e controla os dispositivos de irrigação. O sistema mantém um registro detalhado de todas as ações realizadas, incluindo leituras de sensores, ativações de dispositivos e tempos de espera.

O sistema suporta operações complexas através de operadores lógicos (E, OU) e diferentes tipos de comparações ($<$, $>$, $<=$, $>=$, $==$), permitindo a criação de regras sofisticadas para o controle da irrigação. Por exemplo, é possível criar regras como ”SE SENSOR 1 $<$ 30 E SENSOR 2 $>$ 50 ENTAO LIGAR Bomba”, combinando múltiplas condições.

Todo o processo é acompanhado por um sistema de logs que registra cada ação realizada, facilitando o monitoramento e a depuração do sistema. As mensagens são apresentadas de forma clara e intuitiva, utilizando emojis para melhor visualização do estado do sistema. O tratamento de erros é implementado em todas as etapas, garantindo que o sistema forneça feedback claro e preciso quando encontra problemas, indicando a localização exata do erro e sugerindo correções possíveis. Esta característica é particularmente importante para usuários sem conhecimento técnico, permitindo que eles corrijam seus programas de forma independente.

A arquitetura modular do sistema permite fácil manutenção e extensão, possibilitando a adição de novos recursos ou a modificação de comportamentos existentes sem afetar o restante do sistema. Esta flexibilidade é essencial para adaptar o sistema a diferentes necessidades e configurações de irrigação.

3.2 Analisador Léxico

O analisador léxico é a primeira etapa do processamento da linguagem DSL proposta, sendo responsável por transformar o código-fonte em uma sequência de unidades léxicas (tokens) que serão utilizadas pelas fases seguintes da interpretação. Esta etapa lê o programa linha por linha e identifica padrões pré-definidos como palavras-chave, operadores, identificadores, números e strings, por meio de expressões regulares.

Cada elemento da linguagem, como as palavras ”INSERIR”, ”SE”, ”ENTAO”, operadores de comparação ($<$, $>$, $\leq$, $\geq$, $==$), conectivos lógicos (E, OU), números inteiros e textos entre aspas, é mapeado para um tipo específico de token. O analisador também mantém controle da linha atual do código, facilitando a geração de mensagens de erro mais precisas em etapas posteriores.

A implementação utiliza a biblioteca `re` do Python para realizar a varredura do texto com base em um conjunto de padrões agrupados em uma única expressão regular composta. Cada correspondência encontrada é classificada, e tokens irrelevantes como espaços e quebras de linha são descartados. Os

tokens válidos são então armazenados e encaminhados para a análise sintática.

Essa estrutura permite que a linguagem seja flexível e extensível, possibilitando a adição de novos comandos ou operadores com facilidade, bastando incluir novos padrões na lista de expressões regulares do analisador. Além disso, ao manter os tokens anotados com a linha de origem, o sistema consegue prover mensagens de erro informativas e úteis para o usuário final.

3.3 Analisador Sintático

O analisador sintático é responsável por processar a sequência de tokens produzida pelo analisador léxico e verificar se a estrutura do programa segue as regras gramaticais da linguagem DSL. Essa etapa constrói uma Árvore Sintática Abstrata (AST – Abstract Syntax Tree), que representa de forma hierárquica as instruções válidas contidas no código.

A lógica do parser foi organizada em funções específicas para interpretar os diferentes tipos de comandos da DSL, como declarações de sensores, regras condicionais e comandos de espera. Cada construção sintática esperada é validada de forma sequencial, utilizando um ponteiro de avanço para percorrer a lista de tokens. Caso algum token esperado não seja encontrado na ordem correta, o sistema lança uma exceção indicando um erro de sintaxe.

Para uma declaração de sensor, por exemplo, o parser espera encontrar a palavra-chave ”INSERIR”, seguida de ”SENSOR”, um nome entre aspas, a palavra-chave ”ID” e, por fim, um número inteiro. Regras condicionais iniciam com a palavra ”SE”, seguida por uma comparação entre o valor de um sensor e um limite numérico, podendo conter operadores lógicos como ”E” ou ”OU”. Após as condições, a palavra ”ENTAO” deve anteceder o comando de controle, como ”LIGAR” ou ”DESLIGAR”, aplicado a um dispositivo.

A árvore sintática construída é composta por nós que descrevem a intenção de cada instrução: declaração de sensor, aplicação de regra ou espera por tempo. Cada nó contém os dados extraídos do código, como identificadores, valores numéricos, operadores e alvos das ações.

Essa estrutura intermediária (AST) é essencial para as etapas seguintes de análise semântica e execução, pois fornece uma representação estruturada e validada do programa. O design modular do parser também permite a extensão da linguagem por meio da adição de novos tipos de instruções ou sintaxes alternativas, mantendo a robustez e legibilidade do sistema.

3.4 Analisador Semântico

O analisador semântico é responsável por validar a coerência lógica das instruções representadas na Árvore Sintática Abstrata (AST), assegurando que todas as regras, sensores e ações do programa sejam consistentes, completas e semanticamente corretas. Essa etapa atua como um filtro crítico para evitar erros de interpretação e execução, especialmente em sistemas que envolvem decisões automatizadas.

Durante o processo de análise, são verificados múltiplos critérios. Primeiramente, o analisador garante que todos os sensores utilizados em regras tenham sido previamente declarados, que seus identificadores (IDs) sejam únicos e positivos, e que os nomes estejam devidamente especificados. Para as regras condicionais, verifica-se a validade dos operadores de comparação ($<$, $>$, $\leq$, $\geq$, ==) e dos limites utilizados, que devem estar dentro do intervalo permitido, geralmente entre 0 e 100.

Além disso, as condições compostas com operadores lógicos ”E” (AND) e ”OU” (OR) são verificadas quanto à sua estrutura e validade. As ações associadas às regras devem ser explicitamente ”LIGAR” ou ”DESLIGAR”, e os dispositivos-alvo não podem estar vazios ou indefinidos. Comandos de espera também passam por verificação, exigindo durações numéricas válidas e positivas.

A análise semântica também realiza verificações globais, como a existência de sensores não utilizados, o tempo total acumulado de espera (limitado, por exemplo, a uma hora), e a detecção de regras conflitantes aplicadas a um mesmo dispositivo. Quando duas ou mais regras se aplicam ao mesmo dispositivo com condições incompatíveis ou não complementares, o sistema emite um erro de conflito.

Todos os erros encontrados são registrados com a linha de ocorrência e uma mensagem explicativa, facilitando a identificação e correção por parte do usuário. Ao final da análise, é possível gerar um relatório detalhado contendo todos os erros detectados ou, em caso de sucesso, uma confirmação de que o programa está semanticamente válido. Esta fase é fundamental para garantir que apenas instruções seguras, lógicas e eficazes avancem para a execução.

3.5 Máquina Virtual

A execução dos programas escritos na DSL proposta é realizada por uma máquina virtual interpretativa, desenvolvida com foco em leveza, portabilidade e compatibilidade com computadores e raspberry. Essa máquina virtual opera sobre a Árvore Sintática Abstrata (AST) gerada pelas fases anteriores e é responsável por interpretar e aplicar, em tempo de execução, os comandos definidos pelo usuário.

A máquina virtual mantém registros internos dos sensores declarados, dos dispositivos controlados e de um histórico detalhado de eventos. Quando uma declaração de sensor é interpretada, a máquina armazena seu nome e ID, associando-o a um valor simulado de leitura (por padrão, um valor aleatório entre 0 e 100). Esse valor representa a condição atual de umidade do sensor no momento da execução.

Ao interpretar regras, a máquina verifica se as condições impostas são satisfeitas com base nos valores atuais dos sensores. Essas condições podem ser compostas por operadores relacionais (<, >, <=, >=, ==) e operadores lógicos (`E`, `OU`). Caso as condições sejam satisfeitas, a ação especificada — ligar ou desligar um dispositivo — é executada e registrada.

A máquina também interpreta comandos de espera, utilizando a função `sleep()` para simular o tempo de espera real entre ações. Cada ação, leitura ou transição é registrada em um arquivo de log com data e hora, e o estado final do sistema (valores dos sensores e dispositivos ativos) é salvo em

um arquivo JSON. Isso permite rastrear todo o histórico de execução e facilita a depuração.

O tratamento de exceções também é previsto na execução. Caso sensores não declarados sejam utilizados ou operadores inválidos sejam identificados, a execução é interrompida com mensagens de erro claras e informativas. A robustez dessa arquitetura permite a simulação completa de um sistema de irrigação sem depender de hardware físico, sendo útil tanto em testes quanto em aplicações reais.

A máquina virtual representa, portanto, a camada de execução do sistema, garantindo que os comandos DSL definidos pelos usuários sejam corretamente interpretados, aplicados ao ambiente simulado e integrados à comunicação via MQTT com a IHM e o computador.

4 Resultados

O desenvolvimento do sistema proposto resultou em avanços significativos no controle automatizado de irrigação utilizando uma linguagem específica de domínio (DSL) em português. A principal meta que era permitir que usuários programem sistemas de irrigação de forma acessível e sem conhecimentos técnicos aprofundados, foi alcançada com sucesso. O processo de desenvolvimento foi dividido em módulos, que juntos formaram um fluxo funcional desde a escrita do código até a simulação do controle dos dispositivos.

Dentre os objetivos cumpridos, destacam-se:

- **Criação de uma DSL intuitiva em português:** A linguagem foi desenhada para permitir a definição de regras de irrigação de forma natural, utilizando estruturas como `INSERIR SENSOR`, `SE ... ENTAO ...`, e `ESPERAR`.
- **Interface de programação (Ativar_Sistema.py):** Desenvolvida para interpretar e executar códigos escritos na DSL, realizando análise léxica, sintática e semântica de forma automatizada.
- **Validação completa do código:** O sistema realiza verificações semânticas rigorosas, fornecendo mensagens claras sobre erros e inconsistências, garantindo que apenas instruções válidas sejam executadas.
- **Interface de monitoramento em tempo real com MQTT (sistema_final.py):** Responsável por interagir com sensores e atuadores simulados ou reais, utilizando o protocolo MQTT para comunicação bidirecional.

Apesar desses avanços, algumas funcionalidades desejadas não foram implementadas nesta fase do projeto. Atualmente, após o desenvolvimento do código em português no Ativar_Sistema.py, o usuário ainda precisa replicar manualmente os mesmos parâmetros e configurações no script sistema_final.py para permitir a comunicação via MQTT. A integração entre os dois módulos permanece como uma etapa pendente.

Portanto, a principal limitação atual reside na ausência de uma ponte automatizada entre a interface de programação e a interface de controle em tempo real. A evolução natural do sistema será a implementação de um mecanismo que gere automaticamente, a partir do código em português, os arquivos de configuração ou variáveis necessários para o funcionamento direto da interface MQTT, eliminando a necessidade de intervenções manuais e tornando o processo mais fluido e seguro.

Esses resultados mostram que o conceito é viável e funcional, estabelecendo uma base sólida para futuras expansões com integração completa, suporte a hardware físico e aplicação prática em ambientes reais de produção agrícola.

5 Conclusão

Este trabalho teve como foco o desenvolvimento completo de uma linguagem específica de domínio (DSL), com ênfase na construção de todos os componentes fundamentais de um interpretador: analisador léxico, analisador sintático, analisador semântico e uma máquina virtual interpretativa. A aplicação escolhida serviu como cenário prático para validar o funcionamento da linguagem e demonstrar sua utilidade em um contexto real.

Foram implementados todos os elementos exigidos em projetos de linguagens interpretadas: a análise léxica, responsável pela conversão do código-fonte em tokens; a análise sintática, que valida a estrutura gramatical; a análise semântica, que verifica coerência e segurança das regras definidas e a máquina virtual, que executa os comandos definidos pelo usuário de forma simulada. A integração com interfaces gráficas e o uso de comunicação via MQTT também demonstraram o potencial de aplicação prática da linguagem desenvolvida.

Como resultado, o projeto comprova a viabilidade de se construir uma linguagem personalizada com um fluxo de interpretação robusto e funcional, mesmo quando direcionada a um público não técnico. Além disso, o sistema foi projetado com arquitetura modular e extensível, permitindo que futuras melhorias sejam incorporadas.

Conclui-se, portanto, que os objetivos acadêmicos do projeto foram plenamente atingidos. A linguagem foi definida, validada e executada por meio de uma infraestrutura própria, demonstrando domínio sobre os fundamentos de compiladores e interpretadores. O projeto estabelece uma base sólida para estudos futuros sobre DSLs, ambientes interpretados e sua aplicação em domínios específicos.

References

[1] D. Groeneveld, B. Tekinerdogan, V. Garousi e C. Catal, "A domain-specific language framework for farm management information systems in precision agriculture," *Precision Agriculture*, vol.22, pp.1067–1106, 2021.

[2] D. Spinellis, "Notable design patterns for domain-specific languages," *J. Syst. Softw.*, fev. 2001.

[3] G. Karsai, H. Krahn, C. Pinkernell, B. Rumpe, M. Schindler e S. Völkel, "Design Guidelines for Domain Specific Languages," arXiv, set. 2014.

[4] M. Rouhani et al., "Domain-Specific Languages: A Systematic Mapping Study," *J. Syst. Softw.*, 2015.

[5] TESTEZLAF, Roberto. Irrigação: métodos, sistemas e aplicações. Campinas, SP: Unicamp/FEAGRI, 2017. ISBN 978-85-99678-10-7..

[6] SILVA, Danilo Eduardo Lastória da. Sistema automático de irrigação. Itatiba: Universidade São Francisco, 2010. Monografia (Bacharelado em Engenharia Elétrica – Modalidade Eletrônica) – Universidade São Francisco.

[7] SOUSA, V. F. de; MAROUELLI, W. A.; COELHO, E. F.; PINTO, J. M.; COELHO FILHO, M. A. Manejo da água de irrigação. In: Irrigação e fertirrigação. Brasília, DF: Embrapa, 2011. Cap. 5.
[8] SILVA, Simone Norberto da; NEVES, Eleissandra das. Importância do manejo da irrigação. Enciclopédia Biosfera, v. 17, n. 34, 2020.

6 Acesso do projeto

O repositório do projeto está disponível publicamente no GitHub, contendo o código-fonte, documentação e instruções de uso. O acesso pode ser feito através do seguinte link:

https://github.com/setinwebdesigner/DSL-Irrigacao