XI CONGRESSO DE INICIAÇÃO CIENTÍFICA DO IFSP ITAPETININGA
Itapetininga, 27, 28 e 29 de maio de 2025
Instituto Federal de Educação, Ciência e Tecnologia de São Paulo
Campus Itapetininga

PROJETO E IMPLEMENTAÇÃO DO MECANISMO DE EXECUÇÃO DE BLOCOS FUNCIONAIS DEFINIDOS PELO USUÁRIO CONFORME A ESPECIFICAÇÃO O-PAS

Victoria de Oliveira Spagiari - ITI/IFSP[1]
Prof. Dr. Eduardo André Mossin - IFSP[2]
Prof. Dr. Rodrigo Palucci Pantoni - IFSP[3]

Introdução

Blocos funcionais (*Function Blocks* – FB) são ferramentas fundamentais para a implementação de estratégias e configurações de controle em sistemas de automação industrial. No entanto, a falta de portabilidade entre fabricantes impede que estratégias desenvolvidas em um ambiente sejam reutilizadas em outro (Pantoni et al., 2024). Esse cenário dificulta a modernização e atualização de plantas, além de aumentar custos e a complexidade na integração de tecnologias distintas.

Para superar essas limitações e atender às demandas de tecnologias emergentes, como Inteligência Artificial, Internet das Coisas (*Internet of Things* - IoT) e virtualização de dispositivos, foi criado o padrão O-PAS (*Open Process Automation Standard*), que define uma arquitetura aberta e interoperável, promovendo maior flexibilidade na automação industrial (Qamsane et al., 2022).

O padrão O-PAS contempla blocos funcionais amplamente utilizados, como PID, entrada analógica (*Analog Input* - AI) e saída analógica (*Analog Output* - AO), além de permitir a criação de blocos funcionais definidos pelo usuário (*User Defined Function Blocks* – UDFB). Esses blocos personalizados podem ser exportados e reutilizados em diferentes sistemas compatíveis, independentemente do fabricante, promovendo interoperabilidade e reaproveitamento de soluções.

Estudos anteriores desenvolveram mecanismos para a criação de UDFBs, abrangendo desde a definição lógica até a geração dos arquivos *NodeSet* e *Source Code AddData* (The Open Group, 2023), que estruturam os blocos conforme o padrão O-PAS (Pantoni et al., 2024). No entanto, esses trabalhos se limitaram ao processo de construção da estrutura do bloco, sem avançar para sua execução prática.

Objetivo

O objetivo geral foi desenvolver um sistema capaz de executar UDFBs a partir da lógica em texto estruturado (*Structured Text* - ST) presente no arquivo *Source Code AddData*. Especificamente, buscou-se:

- Converter a lógica dos UDFBs para código C e executar no *runtime* UDFB *Engine*;

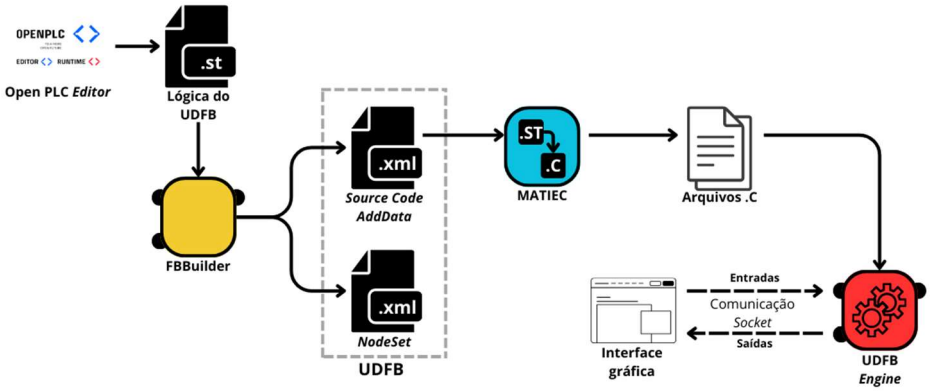
[1]Estudante do Curso de Engenharia Elétrica, IFSP - Sertãozinho. victoria.spagiari@aluno.ifsp.edu.br. Bolsa Captada Externamente (BCE) - APPDI - PD&I Nº 07/2024 - IFSP e Nova Smar S.A. https://orcid.org/0009-0008-4116-0728
[2] Doutor. IFSP - Sertãozinho. emossin@ifsp.edu.br. https://orcid.org/0000-0001-5144-518X
[3] Doutor. IFSP - Sertãozinho. rpantoni@ifsp.edu.br. https://orcid.org/0000-0001-8644-7118

- Desenvolver uma interface gráfica para comunicação e testes de leitura/escrita com o UDFB *Engine*.

Metodologia

Em decorrência do que foi introduzido, este trabalho propõe um fluxo metodológico para converter UDFBs, criados a partir de ferramentas desenvolvidas anteriormente, em código executável compatível com o padrão O-PAS. A Fig.1 apresenta uma visão geral do processo.

Figura 1 - Ilustração do processo de criação e execução de um UDFB.



Fonte: Autoria própria.

A lógica Ladder é inicialmente desenvolvida no OpenPLC Editor (Autonomy, 2022), utilizando tipos de dados O-PAS criados especificamente para este projeto. Como saída, o editor gera um código em ST, que encapsula a lógica do bloco funcional. Esse código é então importado na ferramenta O-PAS Function Block Builder (FBBuilder), desenvolvida por Pantoni *et al*. (2024), onde o usuário define a interface do UDFB — entradas, saídas e parâmetros de configuração/supervisão —, gerando o arquivo *NodeSet* (.xml) conforme a norma O-PAS (The Open Group, 2023). Também é realizado o mapeamento das variáveis ST para os parâmetros O-PAS via o método *AddData*, resultando no arquivo *Source Code AddData* (.xml).

A primeira contribuição deste trabalho ocorre na conversão do código ST (extraído do *Source Code AddData*) para código na linguagem C, utilizando o compilador MATIEC (2001). O código C gerado, correspondente à lógica do UDFB, é então compilado e carregado no *runtime*, desenvolvido em C/C++ e executado em um contêiner Linux.

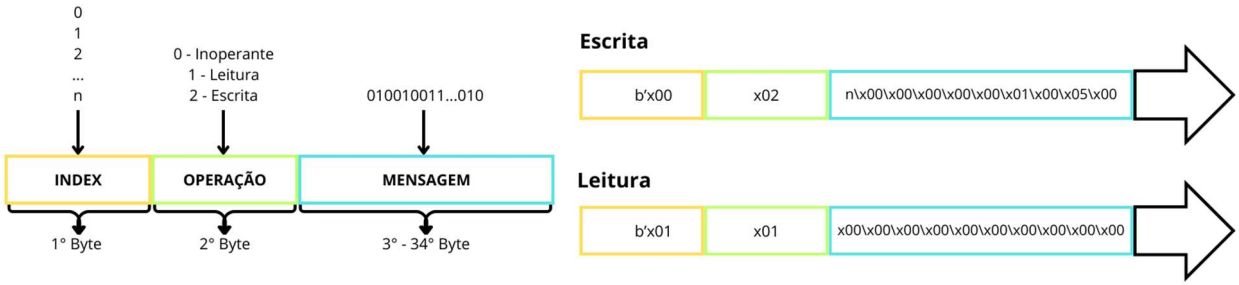
No âmbito deste projeto, atribuímos ao *runtime* o nome UDFB *Engine*, a segunda contribuição inicia-se com a execução do código em tal ferramenta. Originalmente, o UDFB *Engine* suportava apenas tipos padrão, como inteiros, booleanos e reais. Como a norma O-PAS define tipos específicos, foi necessário aprimorá-lo para processar e gerar saídas compatíveis com tipos como *TwoStateDiscrete*, *AnalogUnitRange*, *DataItemTypeBool* e *DataItemTypeReal* (The Open Group, 2023), garantindo conformidade com o padrão e maior flexibilidade na execução dos UDFBs.

O UDFB *Engine* foi projetado para receber valores de entrada, processá-los com base na lógica, originalmente desenvolvida em ST e convertida para C, e retornar as saídas. Para interagir com o UDFB *Engine* e enviar e ler dados para seus parâmetros, uma interface gráfica em Python foi desenvolvida. Esta interface se comunica com o UDFB *Engine* via

socket, utilizando um protocolo desenvolvido especificamente para este projeto. A Fig. 2a ilustra a estrutura desse protocolo: o buffer de comunicação possui três partes — o primeiro byte referência o índice da variável (atribuído sequencialmente ao carregar a lógica), o segundo byte indica a operação (0: nenhuma, 1: leitura, 2: escrita) e a terceira parte, de tamanho variável (1 a 32 bytes), representa o valor da variável. Assim, o buffer pode ter até 34 bytes. Em leituras, o valor retornado na terceira parte é preenchido com zeros. Essa estrutura é usada tanto por cliente quanto por servidor. Exemplos de mensagens de leitura e escrita são apresentados na Fig. 2b.

A interface citada acima é exibida na Fig. 3 (seção de resultados) e constitui outra contribuição deste projeto.

Figura 2 - Protocolo desenvolvido para a comunicação cliente-*engine*.



(a) Estrutura da mensagem (b) Exemplo de mensagens.

Fonte: Autoria própria

Resultados

Para validar o mecanismo desenvolvido, foi criada uma lógica simples contendo uma entrada, uma saída e um parâmetro de configuração. A entrada, denominada *OPAS_Signal_In*, e a saída, *OPAS_Signal_Out*, são ambas do tipo *TwoStateDiscrete*. Além disso, foi definido um parâmetro de configuração do tipo booleano, chamado de *invert*, conforme apresentado no Código 1, retirado do *Source Code AddData* do bloco em questão.

Os sinais da norma O-PAS são estruturas que possuem múltiplos parâmetros. No entanto, para este experimento, foram utilizados valores fictícios para os campos *status*, *timeStamp* e *accessLevel*, pois estes, neste momento não serão utilizados na lógica do UDFB proposto. O comportamento da lógica implementada é o seguinte:

- Se o parâmetro *invert* for 0 (falso), o valor de *OPAS_Signal_In.value* é repassado diretamente para a saída *OPAS_Signal_Out.value*.
- Se *invert* for 1 (verdadeiro), o valor de *OPAS_Signal_In.value* é invertido antes de ser enviado para a saída.

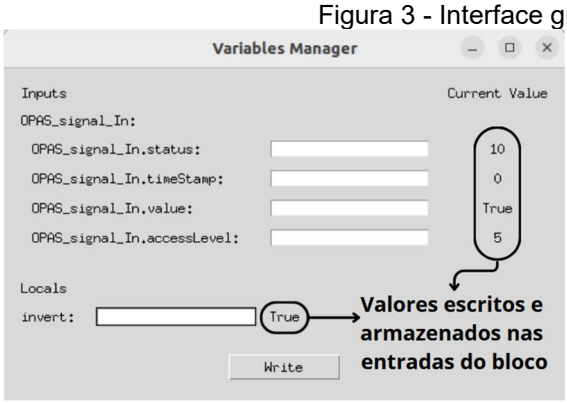
Código 1 - Lógica do UDFB de exemplo retirado do *Source Code AddData*

```
TYPE
 TwoStateDiscrete : STRUCT
        status : DWORD := 1;
        timeStamp : INT := 0;
        value : BOOL := TRUE;
        accessLevel : INT := 0;
 END_STRUCT;
END_TYPE
```

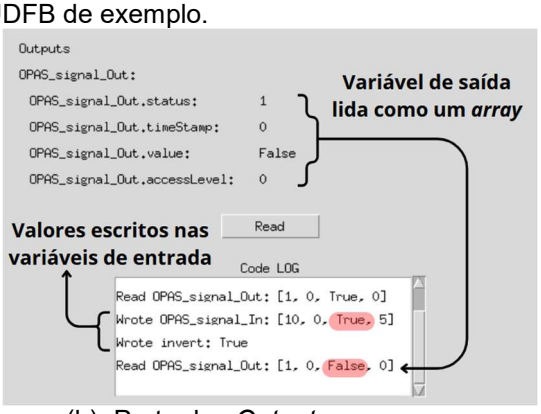
```
…
VAR_OUTPUT
        OPAS_signal_Out :
TwoStateDiscrete;
 END_VAR
 VAR
        invert : BOOL := TRUE;
        _TMP_XOR18_OUT : BOOL;
```

A interface gráfica, com os resultados do processo de escrita e leitura, pode ser observada na Fig. 3. O campo *Code LOG* exibe o resultado das operações realizadas. Nota-se a inversão dos valores do campo *value* (em vermelho) entre a entrada e a saída, uma vez que o parâmetro *invert* está configurado como verdadeiro. Cabe reforçar que a comunicação entre esta interface e o UDFB *Engine* foi realizada através do protocolo apresentado na figura 2.

Figura 3 - Interface gráfica do UDFB de exemplo.



(a) Parte das *Inputs*.



(b) Parte das *Outputs*.

Fonte: Autoria própria

Conclusão

O mecanismo proposto demonstrou-se capaz de executar um UDFB, permitindo tanto a escrita nas variáveis quanto a leitura das saídas dos blocos funcionais. Além disso, foi possível suportar tipos de dados exclusivos da norma O-PAS, como *TwoStateDiscrete*, *AnalogUnitRange*, *DataItemTypeBool* e *DataItemTypeReal*.

A implementação do UDFB *Engine* baseado em C/C++ e a utilização do software MATIEC para converter códigos em *Structured Text* (ST) demonstraram ser abordagens viáveis para garantir a execução dos blocos funcionais. A interface gráfica desenvolvida facilitou a interação com o sistema, permitindo a escrita e leitura das variáveis através da comunicação por *Socket* e de mensagens estruturadas em um formato proposto. Vale lembrar que todo o processo teve início a partir dos arquivos *NodeSet* e *Source Code AddData* , responsáveis por garantir a compatibilidade dos blocos funcionais entre diferentes fabricantes.

Apesar dos avanços obtidos, ainda são necessários testes adicionais para validar melhor o mecanismo e garantir sua robustez em diferentes cenários industriais. Como trabalho futuro, pretende-se integrar o mecanismo desenvolvido ao sistema da empresa financiadora deste projeto, ampliando sua aplicabilidade no contexto industrial.

Agradecimentos

Os autores agradecem à empresa Nova Smar pela colaboração e apoio no desenvolvimento desse projeto (APPDI - PD&I Nº 07/2024 - IFSP e Nova Smar S/A).

Referências

AUTONOMY. OpenPLC Editor 2.01. 2022. Disponível em: https://autonomylogic.com. Acesso em: 26/03/2024.

MATIEC. IEC 61131-3 Compiler. 2001. Disponível em: <https://github.com/nucleron/matiec>. Acesso em: 26/06/2024.

PANTONI, R. P. et al. Design and implementation of o-pas user-defined function blocks. Journal of Electrical Systems and Information Technology, v. 11, p. 55, 2024. Disponível em: <https://link.springer.com/article/10.1186/s43067-024-00183-9>.

QAMSANE, Y. et al. Open process automation- and digital twin-based performance monitoring of a process manufacturing system. IEEE Access, v. 10, p. 60823–60835, 2022.

THE OPEN GROUP. O-PAS™ Standard, Version 2.1. Apex Plaza, Forbury Road, Reading, Berkshire, RG1 1AX, Reino Unido, 2023.