Uma Arquitetura de Middleware Modular e Agnóstica a Protocolos para Sistemas de Gêmeo Digital

Lucas S. dos Santos[1,2], **Luiz C. G. Maria**[2] **e Raphael C. S. Machado**[1]

[1]Instituto de Computação – Universidade Federal Fluminense (UFF)
Niterói – RJ – Brasil

[2]Instituto SENAI de Inovação em Sistemas Virtuais de Produção
(ISI SVP) – Benfica, RJ – Brasil

{lseveriano,raphaelmachado}@ic.uff.br, lgmaria@firjan.com.br

Abstract. *This paper proposes a modular and protocol-agnostic middleware architecture to decouple communication mechanisms from the functional core of the Digital Twin. The proposal was evaluated through a case study based on a training simulator, considering read and write operations. The results indicate that the proposed architecture promotes architectural decoupling without compromising the performance requirements of the simulation.*

Resumo. *Este artigo propõe uma arquitetura de middleware modular e agnóstica a protocolos para desacoplar os mecanismos de comunicação do núcleo funcional do Gêmeo Digital. A proposta foi avaliada em um estudo de caso baseado em um simulador de treinamento, considerando operações de leitura e escrita. Os resultados indicam que a arquitetura proposta promove o desacoplamento arquitetural sem comprometer os requisitos de desempenho da simulação.*

1. Introdução

Sistemas de Gêmeo Digital (*Digital Twin* – DT) têm sido amplamente adotados em cenários industriais e de infraestrutura crítica para representar, monitorar e analisar o comportamento de ativos físicos (*Physical Twin – PT*) por meio de sua contraparte digital [Rodríguez-Alonso et al. 2024]. Os DTs dependem de interações contínuas com o ambiente físico – tanto em termos de aquisição de dados provenientes de sensores quanto o envio de comandos a atuadores – o que requer mecanismos eficientes de comunicação com a finalidade de sincronizar estados entre o mundo físico e o modelo virtual.

Apesar do potencial estratégico dos DTs e do crescimento de sua adoção [MarketsandMarkets 2023], sua implementação ainda enfrenta desafios relacionados à arquitetura, interoperabilidade e comunicação [Almeida et al. 2023]. Em ambientes industriais, a utilização de protocolos especializados, como OPC UA [Mahnke et al. 2009], pode aumentar a complexidade do desenvolvimento e introduzir limitações relacionadas à modelagem de dados, interoperabilidade e manutenção.

Trabalhos recentes têm proposto arquiteturas e *frameworks* para apoiar a implementação prática de DTs [dos Santos et al. 2024, Alvarenga et al. 2024, Rolle et al. 2020]. Entretanto, tais propostas ainda apresentam limitações quanto ao desacoplamento entre mecanismos de comunicação e regras de negócio dos DTs. Em pa-

ralelo, abordagens baseadas em REST e GraphQL têm sido utilizadas para abstrair protocolos industriais específicos [Hietala et al. 2020a, Abdelgawad et al. 2017]. Contudo, essas soluções não foram projetadas para cenários com requisitos mais rigorosos de baixa latência e, ainda, permanecem dependentes de protocolos específicos.

Diante deste cenário, este trabalho propõe uma arquitetura de *middleware* modular e agnóstica a protocolos, projetada para desacoplar os mecanismos de comunicação do núcleo funcional do DT. A arquitetura é responsável por encapsular a implementação do protocolo, o que permite que a lógica de negócio e os modelos do DT evoluam independentemente dos mecanismos e protocolos de redes implementados. Complementarmente, a arquitetura foi projetada com foco em eficiência, buscando reduzir o *overhead* introduzido pela arquitetura e baixo os níveis de latência. Para validação da arquitetura proposta, o estudo de caso Simulador de Treinamento de Imersão para submarinos da classe Tupi (TI) apresentado em [dos Santos et al. 2024] foi utilizado como base para análise experimental do *middleware*. A Seção 2 apresenta os trabalhos relacionados, a Seção 3 descreve a arquitetura proposta, a Seção 4 apresenta a avaliação experimental e, por fim, a Seção 5 apresenta as conclusões e trabalhos futuros.

2. Trabalhos Relacionados

Sistemas baseados em arquiteturas orientadas a serviço (*Service Oriented Architecture* – SOA) têm se destacado pela capacidade de promover interoperabilidade em ambientes distribuídos, especialmente em aplicações de DTs [SOARES 2021]. Em [Grüner et al. 2016], os autores propõem uma extensão da arquitetura OPC UA para embarcar funcionalidades RESTful (*Representational State Transfer*), com o objetivo de melhorar a interoperabilidade entre sistemas industriais e aplicações web. A solução foi implementada na pilha de comunicação do *Open62541*[1]. Nessa perspectiva, o trabalho apresentado em [Schiekofer et al. 2018] segue na abordagem de extensão do protocolo OPC UA baseada na arquitetura REST. Porém, a nova proposta introduz um mecanismo de invocação *session-less* e estabelece um mapeamento entre serviços do OPC UA e métodos HTTP.

Em [Cavalieri et al. 2019], os autores apresentam a *OPC UA Web Platform*, uma plataforma baseada em REST que abstrai a complexidade do OPC UA por meio de uma interface RESTful, simplificando o acesso de aplicações web a sistemas industriais. De forma semelhante, os trabalhos de [Hietala et al. 2020b, Hietala et al. 2020a] propõem uma API baseada em GraphQL para facilitar o acesso a dados de servidores OPC UA por meio de uma interface padronizada. Complementarmente, [Ala-Laurinaho et al. 2022] comparou abordagens REST e GraphQL. O estudo indicou melhor desempenho do GraphQL em operações com múltiplas requisições, enquanto REST apresentou desempenho relativamente superior em operações unitárias.

Em [Künzel et al. 2021] foi proposto o desenvolvimento de um *middleware* modular e flexível para automação industrial, utilizando LabVIEW [Travis and Kring 2006] e OPC UA, com a finalidade de prover comunicação entre sistemas e dispositivos de diferentes fabricantes.

[1]Disponível em `https://www.open62541.org/`

3. Arquitetura do Middleware Proposta

A Figura 1 apresenta a arquitetura do middleware. Nesse sentido, o middleware é posicionada como uma camada intermediária entre o Sistema Físico (PT) e o Sistema Virtual (DT).

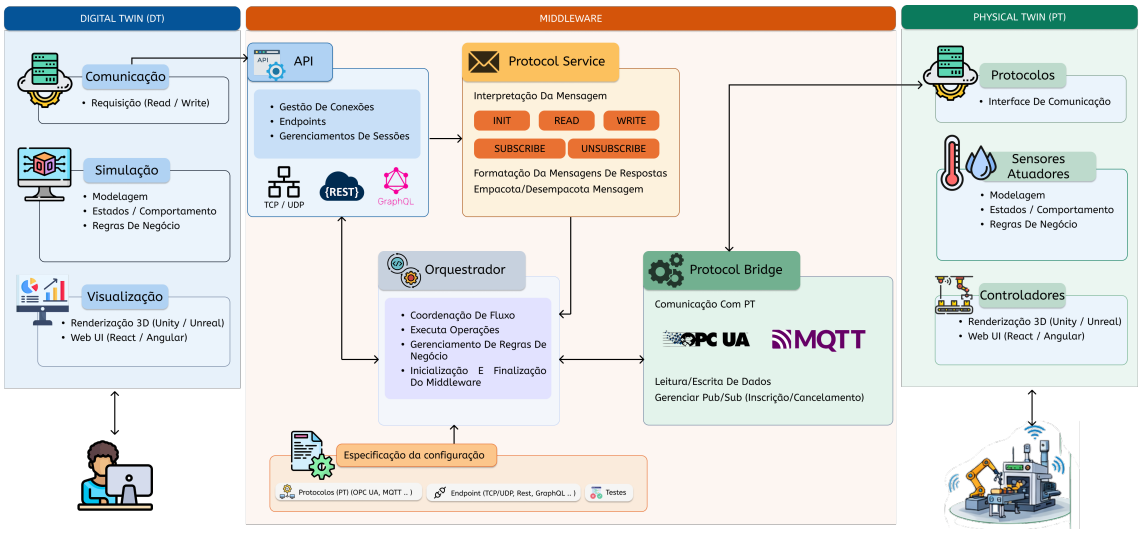

Figura 1. Visão geral da arquitetura do middleware.

O componente **Digital Twin (DT)** representa a contraparte virtual do sistema físico, responsável por modelar seu comportamento, estados e interações. Na arquitetura proposta, o DT executa operações como leitura e escrita de dados e processa as respostas provenientes do PT. Por ser desacoplado dos demais, este componente pode ser implementado utilizando diferentes tecnologias (Unity, Unreal Engine, React, Angular e Vue.js). Em contrapartida, o **Physical Twin (PT)** corresponde ao ativo físico real, dotado de sensores, atuadores e controladores responsáveis pela aquisição e execução de comandos no ambiente físico.

No contexto do middleware, a **API** atua como o ponto de entrada entre o Digital Twin e a infraestrutura interna do sistema. Sua principal responsabilidade é estabelecer e gerenciar o canal de comunicação com o middleware, incluindo a inicialização do ponto de acesso, aceitação de conexões e gerenciamento das sessões de comunicação. O componente **Protocol Service** é responsável por definir e implementar o protocolo de comunicação do middleware. Sua principal função é interpretar as mensagens recebidas do Digital Twin (DT), identificar os comandos nelas contidos e mapeá-los para as operações internas correspondentes. Nesse sentido, o **Protocol Service** atua como uma camada semântica que estabelece a ponte entre o fluxo de dados recebido pela API e os serviços funcionais do middleware. Esse mecanismo de mapeamento entre comandos e funções torna o protocolo extensível, permitindo a inclusão ou modificação de comandos sem impactar os demais componentes da arquitetura.

O componente **Protocol Bridge** é responsável por encapsular e implementar a comunicação direta entre o middleware e o PT. Do ponto de vista da engenharia de software, ele atua como um (*wrapper*), cuja função é traduzir as operações abstratas definidas pelo middleware em interações concretas com o sistema físico, de acordo com o protocolo e a tecnologia adotados pelo sistema físico. Finalmente, o **Orchestrator** atua como

o elemento central de coordenação da arquitetura do middleware, sendo responsável por gerenciar o fluxo principal de execução do sistema.

O middleware foi desenvolvido em Python, utilizando a biblioteca `opcua`[2] para comunicação OPC UA e a biblioteca `Socket`[3] para implementação das interfaces TCP e UDP. O projeto está disponível em `https://gitlab.com/lseveriano/middleware_dt`. Para troca de mensagens, foi adotado o formato JSON (*JavaScript Object Notation*), devido à sua flexibilidade e baixo *overhead* [Bray 2017].

4. Avaliação Experimental

A avaliação experimental reproduziu o cenário do TI, apresentado em [dos Santos et al. 2024]. Nessa configuração, o servidor responsável pela emulação do CLP foi executado em um *Raspberry Pi 4 Model B*, equipado com 4 GB de memória RAM. Para executar o *Digital Twin* (*scripts* de testes) foi utilizado o *MacBook Air M4 16GB RAM - Sistema Operacional macOS Sequoia 15.2*. A interconexão entre os dispositivos foi realizada por meio de um *switch TP-Link LS1005G LiteWave*.

Os testes consideraram cargas de 1, 10, 100, 300 e 500 mensagens, com operações de leitura e escrita em *tags* analógicas do OPC UA, executadas em 100 rodadas. Interfaces REST e GraphQL foram implementadas como *baseline* para comparação com os trabalhos relacionados.

4.1. Resultados e Discussão

A Figura 2 apresenta os resultados das operações de leitura. As interfaces TCP e UDP obtiveram as menores latências médias em todas as cargas avaliadas, com desempenho próximo entre si e crescimento aproximadamente linear. Em contraste, REST e GraphQL apresentaram maior *overhead*, sobretudo nas cargas de 300 e 500 pacotes. Esses resultados indicam que interfaces mais próximas da camada de transporte reduzem a sobrecarga de comunicação e são mais adequadas aos requisitos de desempenho da simulação.

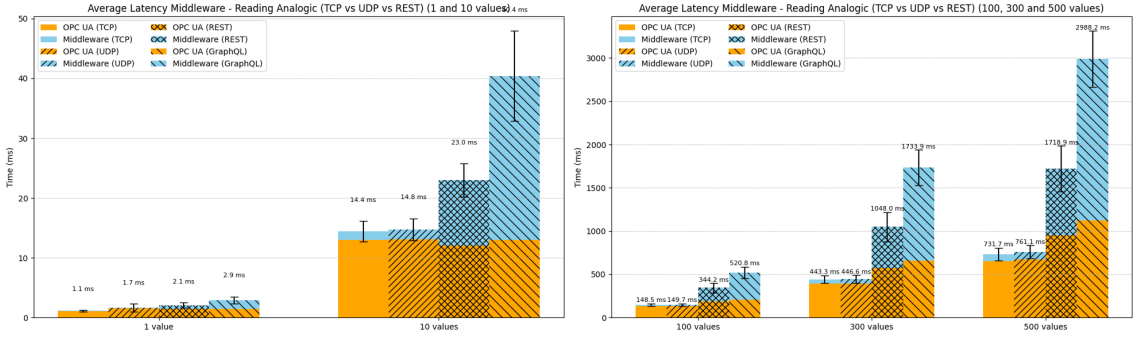

Figura 2. Resultado das operações de leitura.

Os resultados da Figura 3 e Tabela 2 mostram comportamento semelhante ao observado nas operações de leitura. TCP e UDP apresentaram as menores latências, enquanto REST e GraphQL apresentaram maior *overhead*, o que foi mais evidentes em cargas elevadas.

[2]`https://github.com/FreeOpcUa/python-opcua`
[3]`https://docs.python.org/3/library/socket.html`

Tabela 1. Latência média e desvio padrão para leitura.

Prot.	1	10	100	300	500
TCP	1.15±0.13	14.44±1.75	148.54±14.69	443.30±43.80	731.74±75.73
UDP	1.66±0.66	14.76±1.84	149.67±14.69	446.62±43.93	761.11±77.45
REST	2.06±0.42	23.01±2.76	344.18±56.24	1048.02±170.80	1718.88±265.80
GraphQL	2.94±0.58	40.42±7.52	520.77±68.16	1733.87±204.29	2988.16±324.86

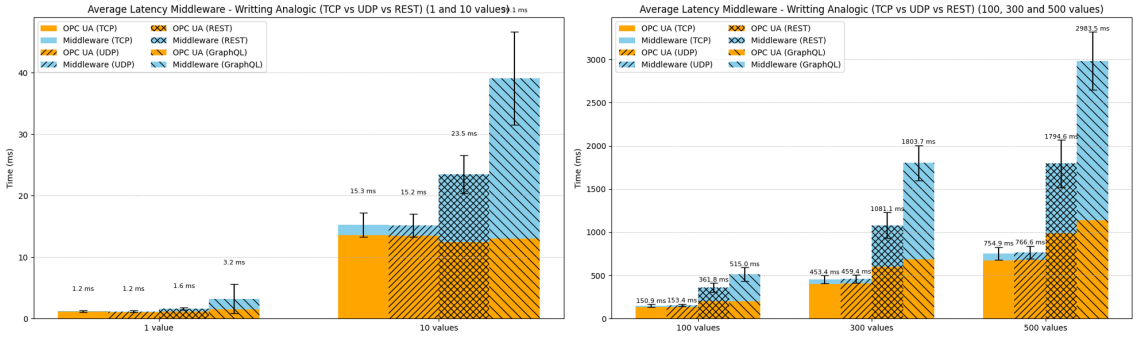

Figura 3. Resultado das operações de escrita.

Tabela 2. Latência média e desvio padrão para escrita.

Prot.	1	10	100	300	500
TCP	1.22±0.15	15.27±1.96	150.90±15.55	453.42±44.83	754.91±74.24
UDP	1.18±0.14	15.17±1.90	153.36±15.17	459.40±45.20	766.60±75.73
REST	1.62±0.17	23.46±3.09	361.82±52.21	1081.07±147.83	1794.63±273.44
GraphQL	3.24±2.36	39.08±7.56	514.98±78.15	1803.70±202.53	2983.51±335.10

5. Conclusão e Trabalhos Futuros

Este trabalho apresentou uma arquitetura de *middleware* modular e agnóstica a protocolos para sistemas de Gêmeo Digital, com foco no desacoplamento entre os mecanismos de comunicação e o núcleo funcional da aplicação. Para validação, foi realizado um estudo de caso baseado no Simulador de Treinamento de Imersão para submarinos da classe Tupi, considerando operações de leitura e escrita sobre *tags* analógicas. Os resultados indicaram que as interfaces TCP e UDP apresentaram consistentemente as menores latências em comparação às abordagens REST e GraphQL, indicando que a proposta foi capaz de preservar requisitos de desempenho sem comprometer o desacoplamento arquitetural.

Como trabalhos futuros, pretende-se avaliar o *middleware* em cenários de comunicação assíncrona e investigar mecanismos de desacoplamento temporal por meio de persistência mecanismos de mensagens e filas. Além disso, serão exploradas estratégias para redução do tamanho das mensagens, como formatos binários, e mecanismos de segurança para proteção da comunicação.

Referências

Abdelgawad, K., Gausemeier, J., Trächtler, A., Gausemeier, S., Dumitrescu, R., Berssenbrügge, J., Stöcklein, J., and Grafe, M. (2017). An application-oriented design method for networked driving simulation. *Designs*, 1(1):6.

Ala-Laurinaho, R., Mattila, J., Autiosalo, J., Hietala, J., Laaki, H., and Tammi, K. (2022). Comparison of rest and graphql interfaces for opc ua. *Computers*, 11(5):65.

Almeida, A., Batista, T., Cavalcante, E., Delicato, F., Motta, R., and Vieira, M. (2023). Middleware for digital twins: A systematic mapping study. In *Proceedings of the 1st International Workshop on Middleware for Digital Twin*, pages 19–24.

Alvarenga, C. d. B. C. S. et al. (2024). Framework para digital twins: integração entre o bim, iot e plataformas em nuvem para projetos de estruturas de edifícios industriais.

Bray, T. (2017). The javascript object notation (json) data interchange format. *RFC Editor*, RFC 8259.

Cavalieri, S., Salafia, M. G., and Scroppo, M. S. (2019). Integrating opc ua with web technologies to enhance interoperability. *Computer Standards & Interfaces*, 61:45–64.

dos Santos, L. S., Maria, L. C., Junior, E. F., and Machado, R. C. (2024). Hybrid simulation framework: Advanced and secure training. pages 13–24.

Grüner, S., Pfrommer, J., and Palm, F. (2016). Restful industrial communication with opc ua. *IEEE Transactions on Industrial Informatics*, 12(5):1832–1841.

Hietala, J., Ala-Laurinaho, R., Autiosalo, J., and Laaki, H. (2020a). Graphql interface for opc ua. In *2020 IEEE Conference on Industrial Cyberphysical Systems (ICPS)*, volume 1, pages 149–155. IEEE.

Hietala, J. et al. (2020b). Real-time two-way data transfer with a digital twin via web interface. Master's thesis.

Künzel, A., Puchta, A., Gönnheimer, P., and Fleischer, J. (2021). Modular and flexible automation middleware based on labview and opc ua. In *IOP Conference Series: Materials Science and Engineering*, volume 1193, page 012109. IOP Publishing.

Mahnke, W., Leitner, S.-H., and Damm, M. (2009). *OPC unified architecture*. Springer Science & Business Media.

MarketsandMarkets (2023). Digital twin market by application, industry, and region - global forecast to 2028. Accessed: 2026-01-21.

Rodríguez-Alonso, C., Pena-Regueiro, I., and García, Ó. (2024). Digital twin platform for water treatment plants using microservices architecture. *Sensors*, 24(5):1568.

Rolle, R., Martucci, V., and Godoy, E. (2020). Architecture for digital twin implementation focusing on industry 4.0. *IEEE Latin America Transactions*, 18(05):889–898.

Schiekofer, R., Scholz, A., and Weyrich, M. (2018). Rest based opc ua for the iiot. In *2018 IEEE 23rd International Conference on Emerging Technologies and Factory Automation (ETFA)*, volume 1, pages 274–281. IEEE.

SOARES, H. D. (2021). Uma arquitetura orientada a serviços de nuvem para apoiar testes experimentais de sistemas ubíquos.

Travis, J. and Kring, J. (2006). *LabVIEW for everyone: Graphical programming made easy and fun (National instruments virtual instrumentation series)*. Prentice Hall PTR.