Modernização de Sistema de Software Embarcado com Arquitetura Hexagonal e Docker: Um Relato da Indústria Solar

Pedro Cobucci
MV Power
Lavras/MG, Brazil
pd.cobucci@gmail.com

Heitor Costa
Universidade Federal de Lavras
Lavras/MG, Brazil
heitor@ufla.br

Resumo

Neste trabalho, é relatada a modernização de um sistema de software embarcado responsável por monitorar usinas solares, com a migração de uma solução restritiva para uma arquitetura robusta e escalável. O sistema legado apresentava alta complexidade ciclomática, baixa manutenibilidade, dependência de um arquivo de configuração JSON extenso, coleta periódica baseada em um *loop* infinito e banco de dados SQLite sem *migrations*, o que resultava em falhas recorrentes e em grande esforço de implantação e manutenção. A refatoração introduziu Programação Orientada a Objetos combinada com Arquitetura Hexagonal, externalização de configurações em YAML (*YAML Ain't Markup Language*), com apoio de uma interface gráfica, *migrations* para o banco de dados SQLite, agendamento de tarefas com Cron e *scripts* de automação de *deploy* usando Docker. Como resultado, o tempo de implantação em novas usinas foi reduzido de horas ou dias para minutos, a complexidade ciclomática média caiu de aproximadamente 8 para 3, e o índice de manutenibilidade aumentou de 49,8 para 63,45, transformando um sistema frágil em uma solução robusta, escalável e alinhada às demandas operacionais do setor elétrico.

Palavras-chave

Refatoração de software, Arquitetura Hexagonal, Sistemas Embarcados, Docker

1 Introdução

Sistemas de software embarcados desempenham um papel crítico na operação de usinas solares, atuando na coleta, no processamento e no envio de dados de campo para a nuvem, mesmo em ambientes com conexão instável e grande heterogeneidade de equipamentos. No contexto do setor elétrico, a disponibilidade e a qualidade desses dados são fundamentais para garantir visibilidade operacional, detectar anomalias e apoiar a tomada de decisão em tempo quase real [4]. No entanto, soluções legadas frequentemente nascem focadas em "fazer funcionar" no curto prazo, acumulando dívidas técnicas que se tornam gargalos de escalabilidade, manutenção e implantação à medida que a quantidade de usinas e dispositivos cresce.

O sistema de software analisado neste trabalho é uma aplicação embarcada da MV Power[1] responsável por monitorar usinas fotovoltaicas, coletando dados de inversores, medidores, relés e sensores e enviando essas informações à nuvem. A versão legada sofria com processos de implantação manuais e demorados, configuração baseada em um único arquivo JSON[2] (*JavaScript Object Notation*) extenso e frágil, ausência de Programação Orientada a Objetos (POO) [3] e de uma arquitetura bem definida, alta complexidade ciclomática, uso de *loop* infinito para coleta periódica de dados e gerenciamento de banco de dados SQLite[3] sem *migrations*. Na prática, isso resultava em um sistema de software instável, difícil de evoluir e com risco elevado de falhas operacionais, justamente em um ambiente em que a robustez é um requisito mínimo.

Neste artigo, é apresentado um relato de experiência sobre a modernização desse sistema de software embarcado, com foco na refatoração [2] arquitetural e na implantação de práticas e tecnologias modernas: POO combinada com a Arquitetura Hexagonal [1], containerização com Docker[4], uso de Cron[5] para agendamento robusto, adoção de *migrations* no banco de dados SQLite, externalização de configuração em YAML[6] (*YAML Ain't Markup Language*) e interface gráfica de apoio, além de mensageria assíncrona via AWS SQS[7] (*Amazon Simple Queue Service*) para comandos remotos. A intervenção resultou em melhorias mensuráveis na manutenibilidade, na complexidade ciclomática e no tempo de implantação, transformando um sistema frágil em uma solução mais robusta, escalável e padronizada.

A estrutura do artigo é a seguinte. Na Seção 2, discute-se o contexto industrial do setor elétrico, os problemas práticos enfrentados com o sistema legado e por que a modernização era crítica. Na Seção 3, apresentam-se os principais resultados qualitativos e quantitativos observados após a refatoração. Na Seção 4, detalham-se as decisões arquiteturais, as tecnologias empregadas e a forma como a solução se integra a cenários industriais semelhantes. Na Seção 5, são descritas as medidas utilizadas para avaliar a modernização: complexidade ciclomática, índice de manutenibilidade, tempo de implantação e robustez operacional. Na Seção 6, são discutidos os desafios enfrentados durante o processo de refatoração, as estratégias de coexistência com o sistema legado e os *trade-offs* das tecnologias adotadas. Na Seção 7, são sintetizados os aprendizados e as contribuições para a prática industrial.

2 Relevância

O sistema de software da MV Power em questão opera em um contexto particularmente sensível: o monitoramento de usinas solares, no qual interrupções na coleta de dados podem comprometer a visibilidade da operação, atrasar a detecção de falhas e impactar indicadores financeiros e regulatórios. Além disso, o ambiente é adverso: conexão de internet instável, tamanhos de planta distintos, grande diversidade de equipamentos com protocolos e unidades

[1] https://mvpower.com.br/
[2] https://www.json.org/json-en.html
[3] https://sqlite.org/
[4] https://www.docker.com/
[5] https://man7.org/linux/man-pages/man5/crontab.5.html
[6] https://yaml.org/
[7] https://docs.aws.amazon.com/AWSSimpleQueueService/latest/SQSDeveloperGuide

de medida diferentes e restrições de hardware nos dispositivos embarcados. Nesses cenários, sistemas legados, projetados para não escalar, tornam-se gargalos estratégicos.

A solução original apresentava um conjunto de problemas típicos de sistemas de software que cresceram sem uma arquitetura robusta: ausência de POO e de Arquitetura Hexagonal, código estruturado como um emaranhado de funções com complexidade ciclomática elevada e forte acoplamento, fluxo de coleta implementado com *loop* infinito e *sleep*, configurações espalhadas em um único arquivo JSON com mais de mil linhas, dependência de ajustes manuais em banco de dados SQLite e implantação feita por *steps* manuais na linha de comando. Na prática, isso significava *deploys* que podiam levar horas ou dias, alta chance de erros humanos, dificuldade para adicionar novos equipamentos ou usinas, instabilidades na execução periódica da coleta e risco de quebra quando qualquer alteração estrutural era necessária.

Do ponto de vista da Trilha da Indústria do SBES, este caso é relevante por tratar de um problema real em um ambiente industrial crítico, com requisitos de confiabilidade e escalabilidade, e por relatar uma modernização que combinou decisões arquiteturais de Engenharia de Software com tecnologias amplamente utilizadas na prática (Docker, Cron, SQS, SQLite, Flask[8], YAML). Trata-se de um exemplo concreto de como aplicar conceitos da área para transformar um sistema de software embarcado legado em um ativo mais alinhado às demandas contemporâneas de operação distribuída e de crescimento do negócio.

3 Impacto e Benefícios

A modernização do sistema de software embarcado na MV Power gerou impactos concretos nas dimensões técnicas e operacionais. Do ponto de vista do processo, o tempo de implantação em uma nova usina foi reduzido, o que antes podia demandar horas ou até dias, envolvendo várias etapas manuais (instalação de dependências, ajustes de rede, configuração de serviços e *cronjobs*) passou a ser realizado em minutos, a partir de *scripts shell* e imagens Docker padronizadas que encapsulam a aplicação e seu ambiente de execução.

No nível de código, a introdução de POO e da Arquitetura Hexagonal reduziu a complexidade ciclomática média dos módulos principais de 8 para aproximadamente 3 pontos, com uma redução expressiva do desvio padrão (de 9,8 para 2,5). Ao mesmo tempo, o índice de manutenibilidade medido por uma ferramenta automatizada (Radon[9]) evoluiu de uma média de 49,8 para cerca de 63,45, indicando um código mais fácil de entender, testar e modificar. A redistribuição de responsabilidades, a centralização das conversões em uma classe de serviço e o uso de configuração declarativa em YAML eliminaram boa parte da duplicação e das condicionais aninhadas responsáveis por "codificar" variações de equipamentos diretamente no código.

Operacionalmente, a troca do *loop* infinito que controlava a coleta por agendamento com Cron aumentou a robustez da execução periódica, falhas pontuais em uma execução não interrompem o agendamento, os horários de coleta tornam-se mais previsíveis e os *logs* são gerados de forma sistemática, facilitando diagnósticos. A integração com AWS SQS[10] agregou a capacidade de receber comandos remotos de forma assíncrona, viabilizando novas automações de operação da usina sem acoplar o sistema de software embarcado à disponibilidade imediata dos serviços na nuvem.

Por fim, a introdução de uma interface gráfica específica para gerar e validar configurações reduziu o risco de erros em arquivos com centenas de linhas, melhorou a experiência dos técnicos em campo e contribuiu para a padronização das configurações entre as usinas. Em síntese, o conjunto de soluções transformou um sistema de software frágil em um sistema de software com base mais robusta, escalável e amigável de operar.

4 Descrição da Aplicabilidade da Solução

A solução é aplicável a cenários em que há sistemas de software embarcados de dados responsáveis por integrar equipamentos de campo a serviços em nuvem, especialmente quando esses sistemas estão em versões legadas e precisam ser modernizados sem interrupção drástica da operação.

Arquiteturalmente, o sistema de software da MV Power refatorado foi organizado segundo o padrão de Arquitetura Hexagonal. No núcleo da aplicação concentram-se as regras de negócio e os modelos (classes) que representam usinas, equipamentos, leituras, comandos e configurações. Em torno desse núcleo, *ports* expõem as operações principais (coleta de dados, envio à nuvem, recebimento de comandos remotos, leitura de configurações), enquanto *adapters* conectam essas portas a tecnologias específicas: *drivers* de equipamentos, protocolos Modbus/TCP, banco de dados SQLite com *migrations*, interface web em Flask, filas SQS e infraestrutura de cron (Figura 1).

A lógica antes codificada em funções específicas de cada equipamento foi movida para uma combinação de: i) arquivos YAML de configuração, nos quais cada atributo de equipamento é associado a endereços de registradores e a expressões (por exemplo, `corrente_c/1000`) e ii) uma classe `ExpressionCalculatorService`, que executa expressões de conversão e normalização de valores. Com isso, a inclusão de novos modelos de inversores ou sensores deixa de exigir a criação de novas funções em Python, passando a ser majoritariamente uma tarefa de configuração declarativa, mais simples de versionar, revisar e replicar entre dispositivos.

No nível de implantação, a adoção de Docker permitiu empacotar o sistema de software e suas dependências em uma imagem padronizada, reduzindo a dependência de configurações específicas de cada dispositivo Linux hospedeiro. *Scripts shell* automatizam a instalação (inclusive do Docker, quando necessário), a construção da imagem, o provisionamento da rede local e o registro de cronjobs para a execução da coleta, da escrita e da eventual reinicialização agendada. Esse mesmo padrão pode ser reaplicado a outras usinas ou a outros tipos de sistemas de software embarcados que demandem replicação em larga escala.

A interface gráfica desenvolvida (usando uma *web stack* típica, como Flask) representa outra dimensão de aplicabilidade, pois reduz a necessidade de manipular diretamente arquivos YAML, oferece testes de comunicação e de leitura/escrita e ajuda técnicos a validar configurações antes de colocá-las em produção. Essa abordagem é

[8] https://flask.palletsprojects.com/en/stable/
[9] https://github.com/rubik/radon
[10] https://docs.aws.amazon.com/AWSSimpleQueueService/latest/SQSDeveloperGuide

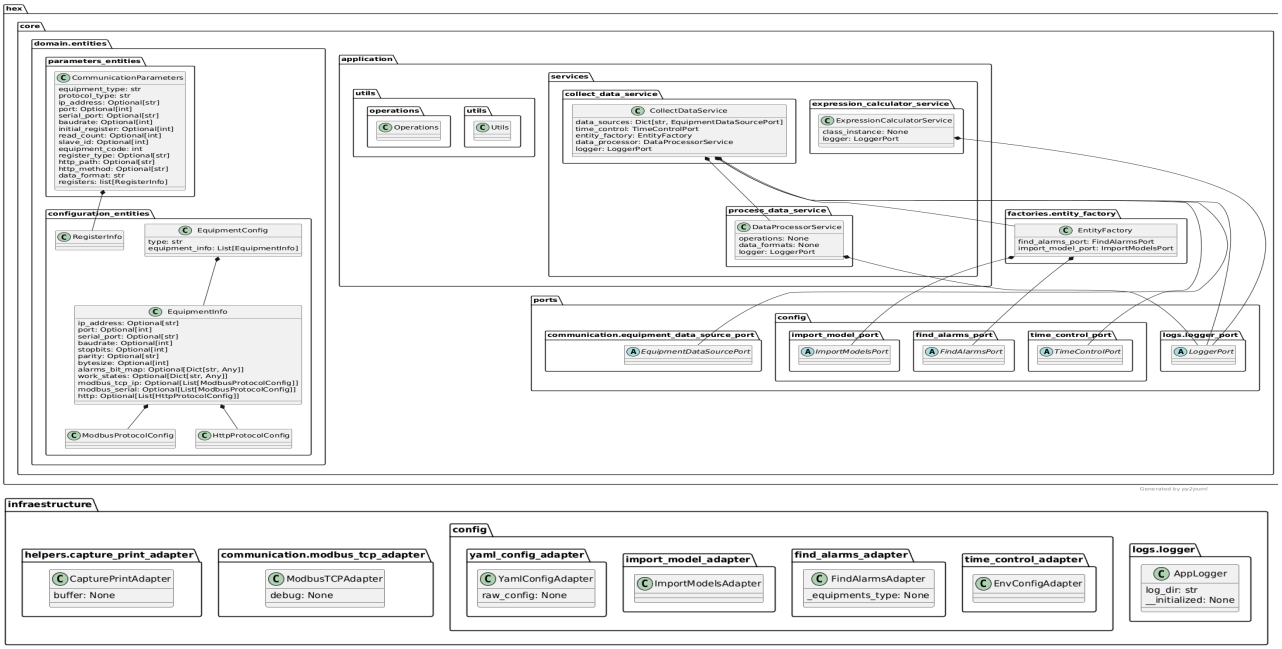

Figura 1: Diagrama de Classes Detalhando a Arquitetura Hexagonal do Sistema de Software.

facilmente replicável em outros contextos, nos quais grande parte da complexidade reside na configuração adequada das conexões e das variáveis dos dispositivos de campo.

5 Indicadores de Sucesso

A avaliação do sucesso da modernização combinou medidas quantitativas do código e do processo com evidências qualitativas do time técnico envolvido. Do ponto de vista quantitativo, foram utilizados:

- **Complexidade ciclomática** calculada por uma ferramenta automatizada, que reflete a quantidade de caminhos independentes em blocos de código. A média dos módulos principais caiu de cerca de 8 para aproximadamente 3 pontos, com forte redução no desvio padrão, indicando módulos mais simples e homogêneos em termos de complexidade, conforme ilustrado na Figura 2;
- **Índice de manutenibilidade** também foi obtido por meio de análise automática, evoluindo de uma média de 49,8 para 63,45, o que sinaliza uma melhoria na legibilidade e na facilidade de modificação do código. Essa evolução pode ser observada no comparativo da Figura 3;
- **Tempo de implantação** medido empiricamente pela equipe, passou de um processo manual que podia consumir horas ou dias (dependendo de ajustes e falhas) para um fluxo automatizado que costuma ser concluído em minutos por usina, com baixa necessidade de intervenção manual;
- **Estabilidade da execução**, embora não tenha sido realizada uma análise estatística formal, observou-se, em ambiente produtivo, uma redução de incidentes relacionados a travamentos na coleta, inconsistências de horário e falhas decorrentes de divergências entre o código e o banco de dados.

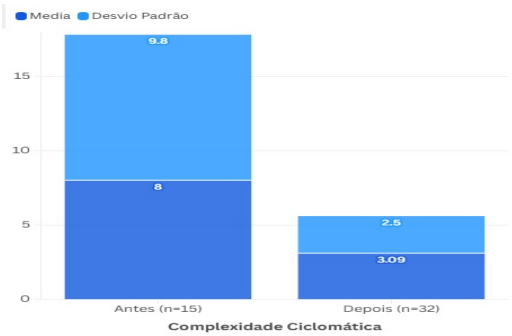

Figura 2: Evolução da complexidade ciclomática média dos módulos.

Entre os indicadores qualitativos levantados junto ao time e à operação, destacam-se:

- Percepção de maior confiança na capacidade do sistema de "aguentar" o crescimento na quantidade de usinas e equipamentos integrados;
- Redução do esforço cognitivo para entender e modificar partes do código, por causa da POO, da Arquitetura Hexagonal e da externalização de lógica de conversão para configuração;
- Melhoria da experiência de técnicos em campo, que passaram a contar com uma interface dedicada para configurar e testar equipamentos, em vez de editar JSONs extensos manualmente.

Como em muitos casos reais na indústria, não se tratou de um experimento controlado, mas o contraste entre o "antes" e o "depois", medido por medidas objetivas e pelos relatos dos envolvidos, fornece evidência robusta do sucesso da intervenção.

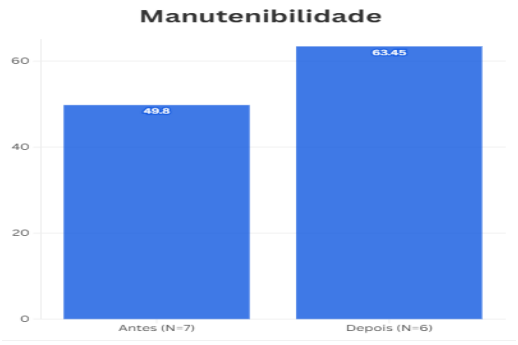

Figura 3: Comparativo do índice de manutenibilidade antes e após a refatoração.

6 Lições Aprendidas e Trade-offs

Apesar dos resultados positivos obtidos com a modernização do sistema legado, o processo de refatoração envolveu desafios, *trade-offs* e aprendizados que merecem ser discutidos para contextualizar a adoção das tecnologias escolhidas em um ambiente. Durante a refatoração, as principais dificuldades encontradas foram o mapeamento e a extração das regras de negócio do sistema legado. Como não havia testes automatizados nem documentação atualizada, muitas variáveis de configuração e regras de conversão de unidades estavam implícitas no código. A migração dessa lógica para arquivos declarativos em YAML e para a classe `ExpressionCalculatorService` exigiu um trabalho exaustivo de engenharia reversa.

Um desafio de grande impacto foi gerenciar o processo de atualização e a operação simultânea dos dois sistemas. Como diversas usinas trabalhavam com o sistema legado e continuariam a utilizá-lo por algum tempo, foi necessário superar dois obstáculos: i) a gestão de contratos de API externa; e ii) a manutenção em "dobro". Para adequar a comunicação entre o software embarcado e a aplicação em nuvem, foi necessário modificar os contratos de API e criar novos *endpoints*, que coexistiram pacificamente com os *endpoints* antigos até a migração total. A exigência de manutenção em "dobro" refletiu-se na velocidade do projeto, pois foi necessário orquestrar o tempo da equipe entre correções emergenciais no sistema legado e o desenvolvimento do novo sistema, o que tornou o processo de refatoração mais lento. A migração ocorreu quando as principais funções do novo sistema estavam adequadas: passou a ser instalada por padrão nas novas usinas e nenhuma atualização foi feita no sistema legado. Quando algum erro ocorria em uma usina antiga, a equipe substituía pelo sistema novo, um processo orgânico que se estendeu até a substituição total.

Por fim, com relação aos *trade-offs* tecnológicos; se por um lado, Docker simplificou a instalação e a configuração do ambiente (mitigando os longos tempos de implantação); por outro lado, foi necessário "sacrificar" capacidade de armazenamento. O tamanho das imagens e a dinâmica dos contêineres exigiram a expansão do armazenamento do *hardware* embarcado utilizado em campo e a criação e implementação de rotinas de "limpeza" periódicas para garantir que o armazenamento não ficasse sobrecarregado.

7 Conclusão

Neste relato, foi apresentada a modernização de um sistema de software embarcado da MV Power utilizado no monitoramento de usinas solares, a partir de uma base legada complexa, frágil e de difícil implantação, para uma solução mais modular, robusta e escalável. As principais ações envolveram a adoção de Programação Orientada a Objetos e de Arquitetura Hexagonal, a introdução de containerização com Docker, o uso de Cron para substituir *loops* infinitos de coleta, a integração de *migrations* ao banco de dados SQLite, a migração de configuração manual em JSON para YAML declarativo, apoiada por uma interface gráfica, e a implementação de mensageria assíncrona via AWS SQS para comandos remotos.

Os resultados indicam os seguintes ganhos: redução significativa da complexidade ciclomática, aumento do índice de manutenibilidade, diminuição do tempo e do esforço de implantação, maior estabilidade na execução periódica e uma base mais adequada para evoluções futuras. Do ponto de vista da Engenharia de Software aplicada à indústria, o caso ilustra como práticas de refatoração, remodularização e rearquitetura, combinadas com tecnologias amplamente disponíveis, podem ser aplicadas a sistemas embarcados reais em operação, sem "parar o mundo".

Como trabalhos futuros, vislumbram-se a adoção sistemática de testes automatizados (de unidade e de integração) para aumentar a segurança das mudanças, a exploração de plataformas de orquestração de *containers* (como Kubernetes[11] ou similares) para gerenciar dezenas ou centenas de dispositivos, e estudos específicos sobre segurança (*hardening*[12], autenticação, criptografia) em cenários distribuídos de usinas remotamente geridas. A experiência relatada pode servir de referência para outras empresas que enfrentam desafios semelhantes na modernização de sistemas legados em contextos industriais críticos.

Disponibilidade de Artefatos

Os artefatos deste trabalho (como código fonte e configurações específicas) são de propriedade da MV Power e não podem ser disponibilizados publicamente por questões de confidencialidade comercial.

Referências

[1] Alistair Cockburn. 2005. Hexagonal (Ports & Adapters) Architecture. https://alistair.cockburn.us/hexagonal-architecture/

[2] Martin Fowler, Kent Beck, and John Brant. 1999. *Refactoring: Improving the Design of Existing Code.* Addison-Wesley, Reading, MA.

[3] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. 1994. *Design Patterns: Elements of Reusable Object-Oriented Software.* Addison-Wesley Professional, Reading, MA.

[4] L. Patan. 2024. Enhancing reliability in distributed systems: A comprehensive approach to telemetry and monitoring. *International Journal of Scientific Research in Computer Science, Engineering and Information Technology* 10, 5 (2024), 1051. https://ijsrcseit.com/index.php/home/article/view/CSEIT241051051/CSEIT241051051/

[11] https://kubernetes.io/pt-br/

[12] Processo de reforçar a segurança de sistemas, redes e aplicações, minimizando sua superfície de ataque