Universidade Federal de Minas Gerais
Escola de Engenharia
Curso de Graduação em Engenharia de Controle e Automação

Aplicação para cálculo de indicadores de performance de malhas de controle de processos industriais

Felipe T. C. Ribeiro

Orientador: Prof. Frederico Gualberto Ferreira Coelho.
Supervisor: Eng. Juliana Vieira Martins

Belo Horizonte, Dezembro de 2020

Monografia

Aplicação para cálculo de indicadores de performance de malhas de controle de processos industriais

Monografia submetida à banca examinadora designada pelo Colegiado Didático do Curso de Graduação em Engenharia de Controle e Automação da Universidade Federal de Minas Gerais, como parte dos requisitos para aprovação na disciplina Projeto Final de Curso II.

Belo Horizonte, Julho de 2014

Sumário

Capítulo 1

Introdução

1.1 Motivação e Justificativa

É incontestável a incessante busca do setor industrial pelo aumento do desempenho de suas unidades operacionais. Para que as empresas envolvidas nesse ramo possam se manter de forma significativa no mercado, é necessária a constante reavaliação de seus processos, bem como de seus custos com energia, perdas e reprocessamento de material. Independentemente do setor de atuação da indústria a ser avaliada, tais fatores possuem extrema importância, devendo ser monitoradas, controladas e projetadas.

Para elucidar, tomemos como exemplo um processo de beneficiamento minério de ferro. Pode-se dividi-lo em: Britagem, peneiramento, moagem, classificação, concentração, espessamento e filtragem. Cada uma dessas etapas é composta por variáveis a serem controladas de forma individual, como níveis de tanque, temperatura e densidade de misturas, velocidades de motores, posições de válvulas, etc. Para estabelecer o comportamento desejado para tais processos, se faz necessária a aplicação de estratégias de controle regulatório, avançado ou de outras naturezas. O conjunto composto por variável a ser controlada e estratégia de controle utilizada é chamado de *malha de controle*.

Implementado um sistema de controle, obtém-se então um processo minimamente adequado aos parâmetros configurados para o contexto operacional escolhido. Entretanto, não menos importante que essa adequação técnica individual estabelecida por esse sistema, são os resultados e efeitos colaterais gerados naquela linha do processo, bem como a avaliação do conjunto em situações não previstas nas estratégias de controle. Dessa forma, o acompanhamento e metrificação do comportamento de sistemas de controle industriais ganha notabilidade dentro do setor industrial.

Para este monitoramento são utilizados softwares baseadis em CPM (*Controller Performance Monitoring*). Eles possuem ferramental para análise temporal das malhas envolvidas, avaliação de parâmetros de controladores, modelagem de processos e medição de KPIs (*Key Performance Indicators*). Buscando este espaço de mercado, a empresa IHM Stefanini desenvolveu o software LOOP, que se enquadra no tipo citado.

Nos últimos meses o time de desenvolvimento da empresa iniciou, juntamente ao time de analistas de controle que utilizam o sistema, um processo de identificação de pontos de melhoria do software LOOP, envolvendo quesitos de interface, regras de negócio, distribuição de responsabilidades entre os módulos do sistema, arquitetura, integração de código e documentação. Dentre os pontos não satisfatórios listados pela equipe, o módulo responsá-

vel pelo cálculo de KPIs foi apontado como crítico, sendo o primeiro a passar por mudanças significativa. O projeto descrito por esta monografia aborda o processo de reconstrução desse módulo.

1.2 Objetivos do Projeto

O objetivo do projeto é a construção de um módulo para cálculo de KPIs, utilizando-se de conhecimentos de Engenharia de software e desenvolvimento de sistemas. Como requisito, o módulo deve executar de forma cíclica a cada hora, se adequar à arquitetura existente do sistema LOOP e possuir um registro de falhas de cada etapa de cálculos realizada.

1.3 Local de Realização

O projeto de fim de curso foi desenvolvido em parceria com a empresa IHM Stefanini, no setor de Discovery da mesma, responsável pelo desenvolvimento, evolução e manutenção da arquitetura e código fonte dos sistemas ali desenvolvidos. O posto de trabalho variou entre as dependências da empresa, e regime de *home office*, conforme necessário.

A empresa realiza projetos e desenvolvimentos de Automação, Elétrica, Montagem, bem como serviços de otimização de malhas, projetos de controle avançado e gestão. Atua em diversas áreas da indústria, como Mineração, Siderurgia & Metais, Papel & Celulose, e Óleo & Gás.

A IHM Stefanini foi criada em 1994 inicialmente com o nome de IHM como uma integradora de sistemas, instrumentação, elétrica e TI Industrial. Com sua expansão, passa a fazer parte do grupo Stefanini em 2015. A partir daí funda o departamento de inovação e começa a atuar na busca por tecnologias disruptivas e elaboração de produtos digitais.

Atualmente a IHM Stefanini é dividida nos seguintes setores técnicos:

- Departamento de TA;
- Departamento de TI Industrial;
- Departamento Inteligência Industrial;
- Departamento de Discovery;
- Departamento de Elétrica;

Este projeto foi desenvolvido no departamento de Discovery que tem como propósito a descoberta sistemática e estruturação de ofertas de alto valor agregado para a empresa. Subdividido em:

- Produtos: Formação de produtos que resolvam problemas reais de clientes finais;
- Coder: Time responsável pelos softwares do setor. Estrutura, desenvolve e mantém códigos-fonte e infraestruturas das demandas do setor.

1.4 Estrutura da Monografia

O trabalho está estruturado em quatro capítulos, sendo o Capítulo 1 o conjunto contextualização, relevância e principais objetivos do projeto como um todo. O Capítulo 2 contempla apresentação e revisão de conceitos básicos importantes para melhor entendimento do projeto. Já o Capítulo 3 traz informações sobre os recursos necessários, metodologia de desenvolvimento e implementação do projeto. Por fim, no Capítulo 4, a apresentação e discussão de dados, bem como sugestões e dificuldades a serem encontradas no projeto.

Capítulo 2

Revisão Literária

Neste capítulo são apresentados os conceitos utilizados neste trabalho baseado na literatura. Na primeira seção são abordadas definições processuais de desenvolvimento de um software. Na segunda são revisados termos de direcionamento técnico para este processo. Na terceira, conceitos de controle e automação importantes, enquanto, na quarta, trabalhos relevantes que abordam assuntos comuns ao projeto aqui relatado.

2.1 Processo de Software

Um processo de software é composto por um conjunto de atividades relacionadas que levam à produção de um produto de software [28]. A utilização deste conjunto de atividades auxilia no aumento da qualidade do produto desenvolvido, bem como sua manutenção. De forma geral, um processo de software é composto de quatro grandes etapas:

- **Especificação de Software:** Devem ser definidas as funcionalidades e limitações de funcionamento do software a ser desenvolvido;
- **Projeto e Implementação do Software:** O software deve ser produzido buscando o atendimento das especificações;
- **Validação de software:** Deve ser validado para garantir o atendimento às demandas solicitadas;
- **Evolução do Software:** Deve ser evoluído e alterado conforme novas necessidades

2.1.1 Modelo em Cascata

Existem diversos modelos de processo de software, dentre eles o modelo cascata, também conhecido como ciclo de vida de software. Este modelo consiste no encadeamento de suas etapas, tendo o início de uma associado necessariamente ao final de outra. São elas:

- **Definição de requisitos:** São definidas restrições e metas do sistema por meio de entrevista com o usuário. Posteriormente são detalhadas a fim de se tornar especificação do sistema;

- **Projeto de Sistema e Software:** Etapa de definição de arquitetura geral do sistema. Envolve identificação e descrição das abstrações, módulos ou componentes, bem como o relacionamento entre eles;
- **Implementação:** O projeto de software é desenvolvido com seus conjuntos ou unidades de programas;
- **Integração e Testes de sistema:** Os módulos individuais do sistema elaborado são integrados e testados em conjunto, buscando assegurar o funcionamento conforme especificado;
- **Operação e manutenção:** O sistema é colocado em uso. São efetuadas possíveis correções de erros não descobertos em etapas anteriores. Expansões e melhorias de implementação das unidades também podem ser realizadas nesta etapa.

Este modelo dá grande importância à documentação gerada ao final de cada uma das etapas, geralmente amarrando o início da etapa seguinte à aprovação destes documentos. Ele também é consistente com outros modelos de processos de engenharia, tornando o processo mais visível para o nível gerencial.

2.1.2 Desenvolvimento Ágil

O desenvolvimento ágil se baseia na entrega de pequenos incrementos do produto de software, disponibilizando versões para o cliente, a cada período de tempo especificado (em geral, duas ou três semanas). Esse envolvimento do cliente durante o processo de construção possibilita o acompanhamento da evolução dos requisitos do sistema. Tem como outra forte característica a utilização de conversas informações, minimização da documentação, gerando mais responsividade do processo a possíveis mudanças nos requisitos do produto de software.

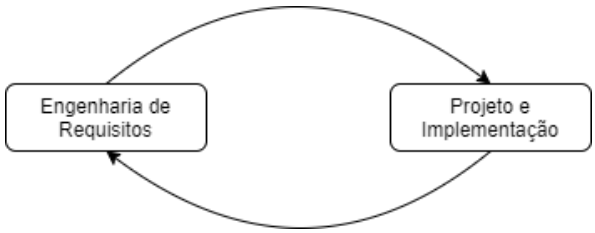

Figura 2.1: Métodos Ágeis

2.2 Arquitetura de Software

Arquitetura de Software é um conjunto de decisões significativas sobre a organização de um sistema de software, a seleção dos elementos estruturais e suas interfaces, a composição destes em subsistemas progressivamente maiores, bem como o estilo de arquitetura que orienta a organização e interação dos módulos envolvidos. A definição da arquitetura de um software passa por etapas e conhecimentos prévios necessários para uma estruturação correta do mesmo [22].

2.2.1 Modelagem orientada a Dados

Modelos construídos sob o paradigma de orientação a dados trazem a sequência de ações envolvidas no processamento daqueles dados, desde sua entrada, até sua saída. Pode ser realizada através de um modelo de sequências[22] destacando a movimentação dos dados daquele módulo ali representado. A figura 2.2 traz um exemplo desse tipo de modelo.

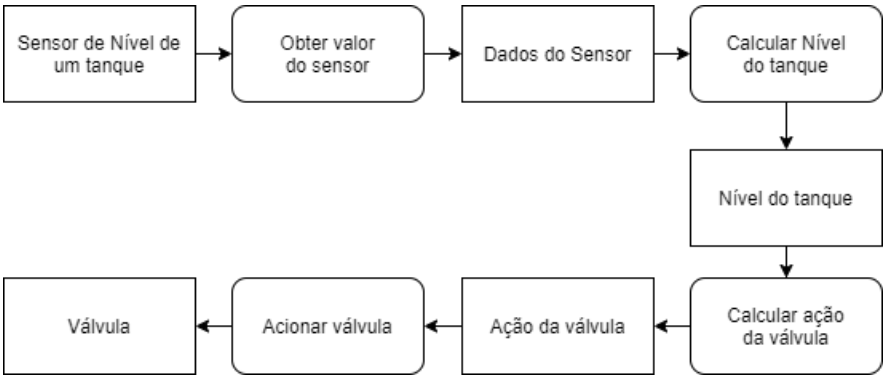

Figura 2.2: Modelo de sequência

2.2.2 Arquitetura em Camadas

Arquitetura que organiza o sistema em um grupo de camadas, onde cada uma delas oferece um conjunto de serviços. Essa noção de separação e independência é fundamental para que haja a possibilidade de alterações localizadas, facilitando as etapas de manutenção e sustentação do produto de software. A figura 2.3 traz uma representação genérica deste tipo de arquitetura.

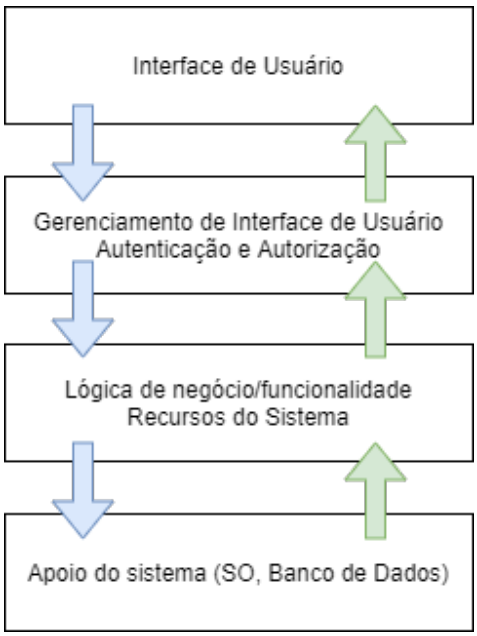

Figura 2.3: Arquitetura genérica em camadas

O relacionamento entre as camadas se dá de forma bem definida. Possui protocolos que indicam como a comunicação entre elas se dá. Uma camada só solicita serviços à inferior, e somente fornece à superior. Esse tipo de divisão facilita e colabora para o processo de desenvolvimento incremental. Uma arquitetura deste tipo muito utilizada e conhecida é a *MVC*, sigla do inglês para *Model-View-Controller*.

2.2.3 Orientação a Objetos

O conceito de OO (Orientação a objetos) consiste na estruturação de um código ou programa utilizando de abstrações para conectar o espaço do problema ao da solução. Os vários objetos construídos nessas abstrações se relacionam através de comandos. Cada um desses objetos possui uma *classificação* que o determina, chamada de *classe*.

Para ficar claro como os objetos se relacionam, os conceitos que seguem são importantes:

- **Encapsulamento:** Estratégia utilizada para garantir que detalhes internos do funcionamento de uma classe. Dessa forma, o conhecimento sobre a implementação interna de uma classe é desnecessária do ponto de vista da instância daquela classe;
- **Herança:** Quando uma classe pode ser um *tipo* de uma outra classe, o conceito de herança pode ser aplicado. Se existe uma classe *veículo* e se deseja construir uma representação *motocicleta*, a segunda (classe filha) pode herdar características da primeira (classe pai), visto que *bicicleta* é um tipo de *veículo*;
- **Polimorfismo:** É o princípio pelo qual duas ou mais classes derivadas de uma mesma classe pai podem invocar métodos que têm a mesma identificação (assinatura) mas comportamentos distintos, especializados para cada classe derivada, usando para tanto uma referência a um objeto do tipo da superclasse.;

2.2.4 Modelo Relacional de Banco de Dados

Banco de dados é uma mecanismo de armazenamento que permite persistência de dados de uma aplicação, programa ou produto de software. O modelo relacional busca o aumento da independência de dados nos sistemas. O Modelo relacional revelou-se ser o mais flexível e adequado ao solucionar os vários problemas que se colocam no nível da concepção e implementação da base de dados. A estrutura fundamental do modelo relacional é a relação. Ela é constituída por um ou mais atributos que traduzem o tipo de dados a armazenar [29].

2.2.5 Containers

Container é uma unidade de software para desenvolvimento e implantação que empacota código e suas dependências para a execução de um aplicativo de forma rápida, escalável e confiável, independentemente de do ambiente de computação que o hospede. No projeto descrito por essa monografia foram utilizados *containers docker* [13].

2.2.5.1 Containter e Máquinas Virtuais

Containers e máquinas virtuais possuem benefícios de isolamento e alocação de recursos, porém contemplam um funcionamento distinto. O primeiro utiliza-se da virtualização do sistema operacional, enquanto o segundo virtualiza o hardware.

Os containers são uma abstração na camada do aplicativo que busca o agrupamento de código e suas dependências. Podem haver múltiplos containers em execução , compartilhando *kernel* [14] do sistema operacional com outros containers, todos isolados em processos diferentes.

Em contrapartida, as máquinas virtuais abstraem o hardware, transformando um servidor em vários servidores. Cada máquina virtual inclui uma cópia de completa do sistema operacional, do aplicativo e suas dependências.

2.2.5.2 Orquestração de containers - Kubernetes

A orquestração consiste na manipulação de containers buscando sua organização, escalabilidade e robustez. Ela permite a automatização de gerenciamento, rede, escala e implantação de softwares construídos sob este paradigma. Sua utilização também permite suporte a estruturas de *DevOps* [8] que integram fluxos de CI/CD (*Continuous Integration/Continuous Delivery*) [7].

2.3 Key Performance Indicators - KPIs

KPIs, sigla do inglês para *Key Performance Indicators*, são indicadores projetados para medir objetivos críticos de um sistema, indo de uma simples malha de controle regulatório, a uma inteira linha de montagem automotiva. O monitoramento desse tipo de indicadores promove o aumento no entendimento do sistema a ser analisado, auxiliando em possíveis tomadas de decisões estratégicas, e aumento de produtividade [27]. O controle desses índices busca geralmente melhorar qualidade e produtividade, alcançando pontos ótimos de operação, bem como aumento da lucratividade da cadeia produtiva em questão. Um exemplo de aplicação de controle de KPIs pode ser visto em [26].

2.3.1 Controller Performance Monitoring - CPM

CPM, sigla do inglês para *Controller Performance Monitoring* é uma técnica das mais usadas para a medição de *KPIs* direcionados a sistemas de automação, tendo T. J Harris [25, 3] como um dos pioneiros no assunto. Sua utilização, nos últimos anos, tem se tornado mais comum como ferramenta para a programas de melhoria de operações promovidas por indústrias e universidades. *CPM* tem como principal objetivo o aumento da confiabilidade de uma planta industrial, sendo um auxiliar para o descobrimento de modos de falhas ocultos, que normalmente não são ligados à sintonia de controle, e sim a outras causas, como defeitos em atuadores (bombas, válvulas) ou formas de operação [31].

2.3.2 KPIs Universais para Plantas Industriais

Como dito, os *KPIs* visam, ao final de seu uso, melhorias nos resultados da empresa que os utilizar, podendo esses estar diretamente (de Negócio) ou indiretamente (de Controle e Processo) ligados aos objetivos da empresa [31]. Os de negócio costumam fornecer medidas valiosas quanto às evoluções ligadas à empresa, tendo caráter de longo prazo. Entretanto, nem sempre estão ligados a ações diretas em uma planta. Lucros, Qualidade e Custos totais estão nessa classificação.

Para que seja possível alcançar sucesso nos *KPIs* de Negócio (longo prazo), é necessária anteriormente a análise de *KPIs* com caráter de curto prazo. Exemplos deste tipo de *KPIs* são: Eficiência, confiabilidade, tempo de acomodação e tempo saída saturada. Esses, por sua vez, podem mais facilmente gerar ações diretas na planta [31].

Desses *KPIs* citados no último parágrafo (e outros presentes em [31]), alguns possuem característica universal, sendo entendidos pelos níveis de negócio e técnicos. Dentre eles, alguns apresentados em [31] se destacam para este tipo de análise:

- **Tempo em modo anormal:** Medição de tempo em que o sistema de controle não está funcionando;
- **Tempo em saturação:** Medição do tempo em que um atuador fica em seu máximo de atuação. Geralmente associado a mal-dimensionamento do atuador;

2.4 Trabalhos Relacionados

Foram encontrados diferentes trabalhos envolvendo *KPIs*: modelagem e controle desses indicadores [30], seu relacionamento com *PSS*, sigla do inglês para *Product-service system* [27], bem como seu uso direcionado a performance de sistemas de controle [31]. Dentre eles, destacou-se o último, devido a relação direta com os índices de performance utilizados para a elaboração do módulo descrito por essa monografia. Os índices citados na seção 2.3.2 são utilizados como base para a construção dos *KPIs* deste trabalho. Na seção **??** serão descritas suas escolhas e elaborações.

Quanto às referências relacionadas aos conceitos de software utilizados para a construção deste trabalho, Lázaro Marques e Dilson Júnior aplicaram o uso de *ORM* [1] para abstração de um banco de dados relacional no trabalho na UNA-SUS/UFMA (Universidade Aberta do SUS/Universidade Federal do Maranhão) [23]. Além disso, José Rocha aponta o uso de arquitetura e desenvolvimento de software orientado a objetos como fator importante para a garantia de qualidade e escalabilidade do um software.

Capítulo 3

Materiais e Métodos

Neste capítulo são descritos os recursos de hardware e software necessários para o desenvolvimento deste projeto. São listados IDE utilizada para desenvolvimento, arquitetura existente a consumir dados da aplicação e escolha de algoritmos de cálculo de KPIs. Também são apresentados os processos de desenvolvimento de software utilizados e funcionamento da equipe dentro da qual foi o projeto foi construído.

3.1 Descrição de Componentes

Esta seção apresenta a arquitetura existente que receberá a aplicação desenvolvida, o Kpi-Executor, bem como os recursos de hardware e software utilizados no seu desenvolvimento.

3.1.1 Software

3.1.1.1 IDE, Linguagem e Banco de Dados

Para o desenvolvimento da aplicação foi escolhida a linguagem *Python* [18], em sua versão 3.8, devido a seu caráter de alto nível e sua grande gama de bibliotecas disponíveis. Para a codificação foi escolhida a IDE *Visual Studio Code* [19], devido à sua multiplicidade de extensões e também por já ser amplamente utilizada pela comunidade de desenvolvedores de software.

Ambas, linguagem e IDE, tem característica *opensource*, sendo ativamente utilizadas e exploradas dia a dia por sua ampla comunidade, além de terem seus módulos e extensões gratuitos.

Quanto ao armazenamento dos dados, foi utilizada uma estrutura relacional de banco de dados, considerando sua praticidade, bem como a estrutura existente descrita nas próximas seções. Foi utilizada uma instância de um banco de dados *PostgresSQL* [20] , uma ferramenta gratuita para a persistência dos dados gerados pela aplicação.

3.1.1.2 Orquestradores e Containers

Como a aplicação desenvolvida foi acoplada a um sistema existente, foi necessário que ela respeitasse o conjunto anteriormente elaborado, bem como se adequasse à arquitetura em nuvem [24] já em funcionamento. Para tal, essa aplicação deveria possuir certo encapsulamento.

Uma forma amplamente utilizada para o encapsulamento e execução de aplicações de grande porte é o formato de *containers*. É uma abordagem de desenvolvimento de software onde um serviço ou aplicativo é empacotado com suas dependências e configurações [15]. Dessa forma, são facilitados os processos de Testes de unidade e integração [28], bem como etapas de evolução [28] e versionamento daquele conjunto. Para o processo de construção de *container* do módulo *Kpi-Executor* foi utilizada a plataforma *Docker* [16], gerando uma imagem de um SO (Sistema Operacional) Linux contendo as dependências e instalações necessárias para a execução do *Kpi-Executor*.

Uma vez empacotada a aplicação, é necessário utilizar uma ferramenta para operar esse pacote, o executando sempre que necessário. Esta utilidade é chamada de *orquestrador*. A ferramenta deste tipo utilizada foi o *Kubernetes* [17]. Este tipo de ferramenta possibilita a automação de operações *containers*, como implantações ou atualizações das aplicações, o gerenciamento de serviços de forma declarativa garantindo a repetibilidade do sistema, a separação de *containers* em diferentes *hosts*, bem como a verificação de integridade e autorrecuperação das aplicações [21].

3.1.2 Estrutura Existente

A aplicação construída funcionará como um módulo de um sistema de CPM existente chamado LOOP. Ele é composto por um conjunto de módulos, cada um com sua responsabilidade bem definida. A figura 3.1 traz o esquema arquitetural do sistema, bem como o relacionamento entre os módulos.

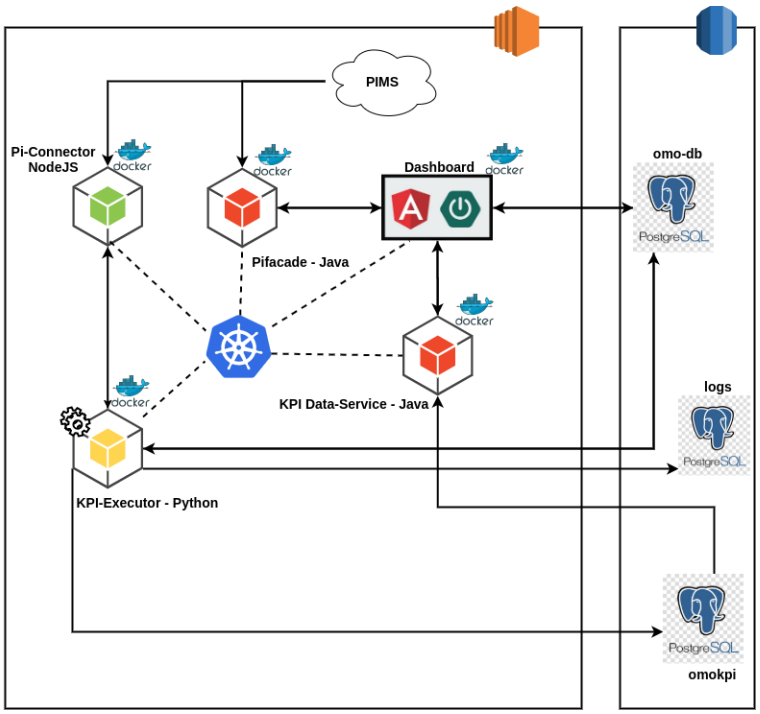

Figura 3.1: Arquitetura Loop

Neste esquema são identificadas as linguagens e tecnologias de cada módulo. São eles:

- **Dashboard:** Aplicação web acessível ao usuário do sistema CPM LOOP composta por um *backend* em *Spring Boot* [11] e *frontend* em *AngularJS* [5];
- **Bancos de Dados:** São três instâncias de banco de dados do sistema. A nomeada como *omo-db* carrega as informações de cadastro e configuração das entidades do LOOP. A nomeada *omokpi* armazena os dados calculados pelo Kpi-Executor, enquanto a *logs* registra os logs de funcionamento da aplicação de cálculos. Todas as instâncias são do tipo PostgreSQL [20];
- **Data-Service:** Módulo do tipo API construído em *Spring Boot* [11] responsável por prover dados registrados no banco *omokpi* à aplicação web *Dashboard*. Formata pesquisa e retorna dados sob demanda;
- **PIMS:** Um sistema PIMS da *OsiSoft* chamado *PI System* [10]. Nele são armazenados os dados de processo dos clientes que utilizam o sistema LOOP. Estes dados são utilizados na realização dos cálculos (Kpi-Executor);
- **Pi-Facade:** Módulo do tipo API construído em *Spring Boot* [11] responsável por prover dados registrados e cadastrar novas informações no PIMS sob demanda da aplicação web *Dashboard*;
- **Pi-Connector:** Módulo do tipo API construído em *NodeJS* [9] responsável pelo fornecimento de dados registrados no PIMS à aplicação de cálculos (Kpi-Executor);
- **Kpi-Executor:** Módulo desenvolvido descrito por esta monografia. Construído em *Python 3.8* [18] e responsável por calcular os índices de performance (KPIs) do sistema LOOP. Opera de forma cíclica, realizando seus cálculos a cada hora.

3.1.3 Hardware

Para realização do desenvolvimento do Kpi-Executor, foi necessário um notebook com 16Gb de memória e processador intel i7. Também foi necessário um ambiente de Testes e validação semelhante ao de produção. São máquinas hospedadas na *Amazon Web Services* [6].

3.2 Metodologia

Para a construção dessa aplicação foi utilizado o um processo de desenvolvimento de software composto por características de um cascata e um incremental [28]. Como o projeto foi entendido como uma demanda associada ao fluxo de trabalho da equipe de Coder, descrita na seção 1.3, ele estava submetido a um fluxo de *Kanban*, o que aponta para o quesito de entrega incremental quando associado ao sistema LOOP como um todo. Entretanto, como se tratava de uma tarefa não pequena, foram necessárias etapas que coincidem com o processo cascata. Neste capítulo serão listadas as etapas de projeto, desenvolvimento e testes do Kpi-Executor, bem como as escolhas relacionadas aos cálculos por ele realizados.

3.2.1 Processo de Desenvolvimento

O processo de desenvolvimento da aplicação iniciou-se com um escopo relativamente simples. Ela deveria realizar de forma cíclica um conjunto de cálculos de índices de performance que são utilizados pelo sistema CPM LOOP, registrá-los em um banco de dados existente, com estrutura também definida, e possibilitar um rastreio de falhas no fluxo de cálculos, identificando o qual tipo de erro, e em que momento este acontecera. Tendo essas informações, ainda foi eram necessárias mais definições de casos específicos não mapeados pela descrição inicial.

3.2.1.1 Levantamento do Requisitos

Considerando a necessidade de um mapeamento detalhado dos casos de cálculo, foi realizado um processo de entrevista com o PO (*Product Owner*) do sistema LOOP, a fim de elucidar o fluxo de cálculo em seu estado ideal, possíveis tratativas em casos não usuais e demais requisitos funcionais e não-funcionais. Deste levantamento, foi elaborado o diagrama de sequência (2.2.1) tratando o fluxo a ser executado pelo Kpi-Executor, exibido na figura (3.2).

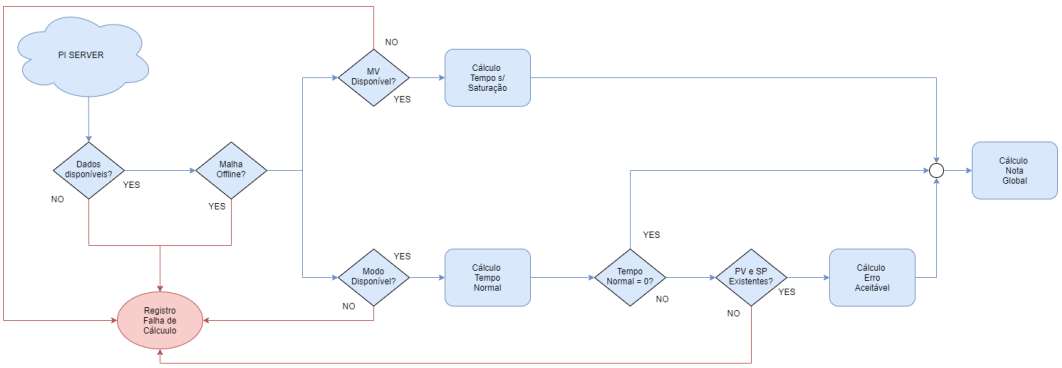

Figura 3.2: Diagrama de Sequência - Kpi-Executor

Após construção deste diagrama de alto nível, foi realizada o nova reunião o validando como fluxo correto de cálculos, incluindo as tratativas de casos específicos. Assim pôde-se caminhar para a próxima etapa do processo.

3.2.1.2 Projeto da Aplicação

Tendo requisitos funcionais e não-funcionais especificados, foi possível então o início do projeto da aplicação. Inicia-se pela definição arquitetural do módulo Kpi-Executor. Foi escolhida uma arquitetura em camadas (seção 2.2.2) devido à familiaridade da equipe com o tipo de construção e a seu caráter de atuação sob demanda e persistência em banco de dados.

Para persistência dos dados em banco, foi escolhida a tecnologia SQLAlchemy [4], amplamente utilizada para acesso via Python a bancos de dados relacionais. Ela possibilita uma abordagem de orientação a objetos (seção 2.2.3) em sua utilização.

3.2.1.3 Desenvolvimento da Aplicação

3.2.1.3.1 Classes auxiliares

Considerando a estrutura do sistema da empresa parceira (Loop) no momento do desenvolvimento do módulo tratado por essa monografia, era tido como de suma importância a possibilidade de utilização de diferentes bancos de dados relacionais, como Microfost SQL Server [12] e PostgresSQL [20]. Assim, foi necessária a utilização da uma *ORM* (Object Relacional Mapper) [1] *SQLAlchemy* [4], possibilitando que o módulo em questão apresentasse características agnósticas a banco de dados. Além da facilitação no processo de substituição de bancos de dados, devido à objetificação das *queries* adiciona segurança ao conjunto do sistema, impedindo tentativas de invasão do tipo *SQL Injection* [2].

Uma vez levantados os pontos citados, foram utilizados os conceitos de orientação a objetos para o desenvolvimento (2.2.3) para a elaboração de duas classes que juntas tornariam o processo de construção das consultas e inserções a banco de dados mais práticas e robustas:

- **Database:** Classe que contempla métodos para conexão e desconexão com o banco de dados, bem como as informações de conexão, como *host*, nome do banco, usuário, senha e *driver*. Seu uso consiste na representação de um banco de dados como um objeto. O *SQLAlchemy* [4] faz gerenciamento automático de múltiplas conexões, possibilitando o instanciamento de diversos objetos dessa classe;
- **Repository:** Classe que contempla os métodos básicos relacionados a consultas, inserções, atualizações e deleções. Seu uso consiste em tê-la como pai para cada nova classe com funções de acesso a dados. Possui um atributo do tipo *Database* que aponta para o banco de dados a ser utilizado.

Uma vez elaboradas tais classes auxiliares, tem-se a etapa seguinte que contempla o conceito de divisão de camadas (2.2.2), novamente orientação a objetos (2.2.3) e modelagem orientada a dados (2.2.1).

3.2.1.3.2 Divisão de Responsabilidades

Considerando uma arquitetura de camadas, o Kpi-Executor foi modularizado dividindo as responsabilidades entre estes módulos. Foram separadas entre os componentes as tarefas de

se consumir e escrever em diferentes bases de dados, operar o fluxo de cálculos, registrar exceções e falhas de cálculo, etc. A figura 3.3 traz o diagrama de componentes que descreve tal modularização.

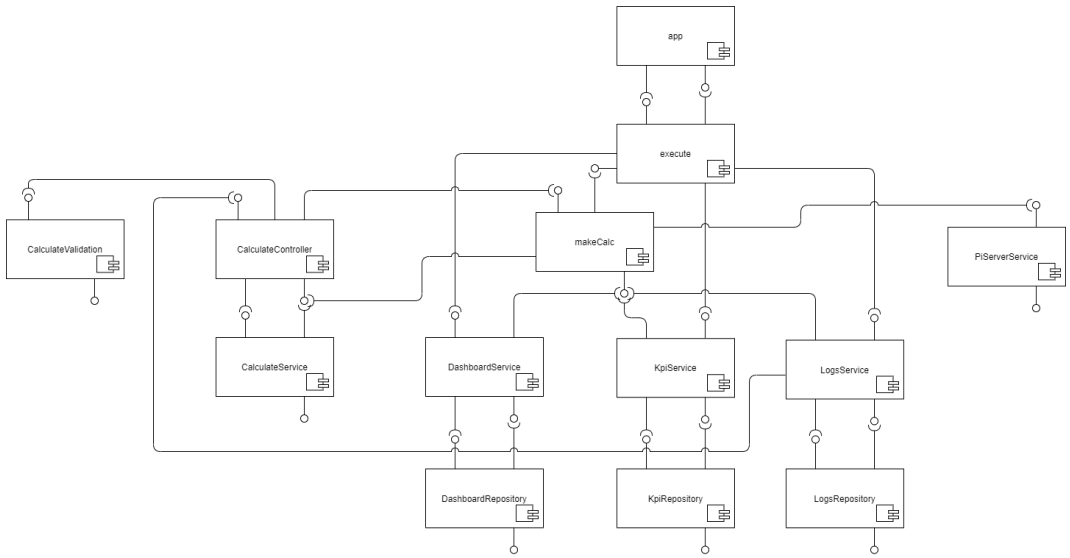

Figura 3.3: Diagrama de Componentes - Kpi-Executor

Cada componente descrito no diagrama é composto por um arquivo independente em Python [18], cada um com suas responsabilidades.

- **app:** Arquivo principal. Ele é o módulo invocado pelo orquestrador Kubernetes [17]. Responsável pela configuração de logs de manutenção e carregamento de variáveis de ambiente do sistema, configurando a aplicação para sua forma de execução local, em ambiente de validação ou produção;
- **execute:** Responsável por instanciar os objetos de banco de dados utilizados pelos próximos componentes. Também invoca DashboardService para montar o objeto com os dados de cadastro para as etapas de cálculos de KPIs;
- **makeCalc:** Opera o fluxo principal ta aplicação. Percorre o objeto de dados de cadastro fornecido pelo componente *execute* e solicitando, a cada iteração, ao *PiServerService* dados de processo cadastrados no PIMS, malha a malha;
- **DashboardService:** Possui regras de negócio relacionadas ao banco *omo-db*. Formata os dados lidos e para inserção neste banco. Também possui os métodos para instanciar os objetos de banco de dados e para solicitar escrita no banco de dados;
- **DashboardRepository:** Possui os métodos correspondentes às queries de inserção e consulta de dados em banco. Tem relação de herança [28] com a classe *Repository* (seção 3.2.1.3.1);
- **PiServerService:** Contempla regras de negócio relacionadas à consulta de dados no PIMS. Formata e realiza requisições *Http* para o módulo externo *Pi-Connector* (figura 3.1). Também possui rotinas para tratamento das respostas com os dados do PIMS;

- **KpiService:** Contempla as regras para formatação de dados calculados para inserção no banco *omokpi*;
- **LogsService:** Possui métodos para conexão com o banco de dados de logs de controle;
- **LogsRepository:** Possui métodos relacionados a inserção e leitura em dos logs em banco de dados. Tem relação de herança com a classe *Repository* (seção 3.2.1.3.1);
- **CalculateController:** Componente chave para o funcionamento da aplicação. Contempla o fluxo descrito pelo levantamento de requisitos na figura 3.2. Também invoca os módulos *KpiService* e *DashboardService* para registro de dados calculados e *LogsService* para persistência dos logs de controle;
- **CalculateService:** Possui os métodos para a realização dos cálculos dos índices de performance. É invocado pelo *calculateController*;
- **CalculateValidation:** Possui métodos auxiliares para a validação do fluxo de cálculos. É invocado pelo *CalulateController*;

3.2.1.3.3 Índices de Performance - KPIs

Explicar os cálculos realizados e citar a seção (ainda não construída) da revisão bibliográfica sobre os índices de performance.

3.2.1.3.4 Registro de Logs de Cálculo

Adicionar aqui o processo de estruturação dos logs de cálculo em banco de dados. Associar esse processo ao fluxo da seção de levantamento de requisitos.

3.2.1.4 Testes e Implantação

Explicar etapas de teste com dados de cliente e validação de fluxo e resultados de cálculos dos índices de performance

Referências Bibliográficas

[1] ORM: Object relational mapper. URL https://www.devmedia.com.br/orm-object-relational-mapper/19056.

[2] SQL injection. URL https://www.devmedia.com.br/sql-injection/6102.

[3] Assesment of Control Loop Performance, language =.

[4] Página Oficial SQLAlchemy. URL https://www.sqlalchemy.org/.

[5] Página Oficial AngularJS. URL https://angularjs.org/.

[6] Página Oficial AWS. URL https://aws.amazon.com/pt/.

[7] Integração e entrega contínuas: pipeline ci/cd. URL https://www.redhat.com/pt-br/topics/devops/what-is-ci-cd.

[8] O que é devops? URL https://www.redhat.com/pt-br/topics/devops.

[9] Página Oficial NodeJS. URL https://nodejs.org/en/.

[10] Página Oficial PI System. URL https://www.osisoft.pt/pi-system/.

[11] Página Oficial Spring Boot. URL https://spring.io/projects/spring-boot.

[12] Microsoft SQL Server. URL https://www.microsoft.com/pt-br/sql-server/sql-server-downloads.

[13] What is a container?, . URL https://www.docker.com/resources/what-container.

[14] O que é kernel?, . URL https://www.tecmundo.com.br/macos/1636-o-que-e-kernel-.htm.

[15] Introdução aos containers e ao Docker, 2018. URL https://docs.microsoft.com/pt-br/dotnet/architecture/microservices/container-docker-introduction/.

[16] Página Oficial Docker, 2019. URL https://www.docker.com/.

[17] Página Oficial Kubernetes, 2019. URL https://kubernetes.io/pt/.

[18] Página Oficial Python, 2019. URL https://docs.python.org.

[19] Página Oficial Visual Studio Code, 2019. URL https://code.visualstudio.com/.

[20] Página Oficial PostgreSQL, 2020. URL https://www.postgresql.org/.

[21] Kubernetes e a tecnologia de containers, 2020. URL https://www.redhat.com/pt-br/topics/containers/what-is-kubernetes.

[22] G. BOOCH, J. RUMBAUGH, and I. JACOBSON. *UML, Guia do Usuario*. 2ª edition, 2005.

[23] L. H. C. M. e Dilson José L. R. Júnior. Aplicação do ORM Doctrine para abstração de banco de dados no desenvolvimento de software em PHP na UNASUS/UFMA.

[24] A. G. Hospedagem Cloud, 2019. URL https://www.hostinger.com.br/tutoriais/o-que-e-hospedagem-cloud/.

[25] T. J. Harris. Recent Developments in Controller Performance Monitoring and Assessment Techniques.

[26] H. Luo, S. X. Ding, A. Haghani, H. Hao, S. Yin, and T. Jeinsch. Data-driven design of kpi-related fault-tolerant control system for wind turbines. In *2013 American Control Conference*, pages 4465–4470, 2013. doi: 10.1109/ACC.2013.6580528.

[27] D. Mourtzis, S. Fotia, and E. Vlachou. Lean rules extraction methodology for lean pss design via key performance indicators monitoring. *Journal of Manufacturing Systems*, 42:233 – 243, 2017. ISSN 0278-6125. doi: https://doi.org/10.1016/j.jmsy.2016.12.014. URL http://www.sciencedirect.com/science/article/pii/S0278612516301017.

[28] I. SOMMERVILLE. *Engenharia de Software*. 9ª edition, 2011.

[29] O. K. TAKAI. *Introdução a Banco de Dados*. 2005.

[30] C. Wang and S. Zhou. Control of key performance indicators of manufacturing production systems through pair-copula modeling and stochastic optimization. *Journal of Manufacturing Systems*, 58:120 – 130, 2021. ISSN 0278-6125. doi: https://doi.org/10.1016/j.jmsy.2020.11.003. URL http://www.sciencedirect.com/science/article/pii/S0278612520301904.

[31] J. Zevenbergen, J. Gerry, and G. Buckbee. AUTOMATION KPIs CRITICAL FOR IMPROVEMENT OF ENTERPRISE KPIs. page 4.