DAS Departamento de Automação e Sistemas
CTC Centro Tecnológico
UFSC Universidade Federal de Santa Catarina

Simulador de Processos e Controle Preditivo para a Indústria de Petróleo e Gás

Relatório submetido à Universidade Federal de Santa Catarina
como requisito para a aprovação da disciplina:
DAS 5511: Projeto de Fim de Curso

Rodrigo da Silva Gesser

Florianópolis, Agosto de 2016

Simulador de Processos e Controle Preditivo para a Indústria de Petróleo e Gás

Rodrigo da Silva Gesser

Esta monografia foi julgada no contexto da disciplina

DAS 5511: Projeto de Fim de Curso

e aprovada na sua forma final pelo

Curso de Engenharia de Controle e Automação

Prof. Daniel Martins Lima

_____

Banca Examinadora:

Prof. Julio Elias Normey Rico
Orientador na Empresa

Prof. Daniel Martins Lima
Orientador no Curso

Prof. Hector Bessa Silveira
Responsável pela disciplina

Prof. José Dolores Vergara Dietrich, Avaliador

Amadeu Plácido Neto, Debatedor

Pedro Augusto Ceriotti, Debatedor

Agradecimentos

Ao Professor Julio Elias Normey-Rico pela orientação e confiança no meu trabalho, dando a atenção e o conhecimento necessário.

Ao Professor Daniel Martins Lima que trabalhou continuamente comigo e teve papel fundamental no decorrer do projeto e seu desenvolvimento.

À ANP e ao PRH-34 pela oportunidade oferecida e o apoio ao longo do projeto.

À minha família, amigos e namorada pelo carinho incondicional.

Resumo

A análise e identificação de processos industriais melhora a percepção do engenheiro acerca das ações que devem ser tomadas em diversos cenários na indústria. Na área da engenharia de controle, o entendimento do processo é essencial para que a sintonia e configuração de controladores tenham resultados satisfatórios, porém a análise pode ser dificultada caso o processo seja muito complexo e também caso controladores avançados sejam utilizados, tornando essa sintonia dispendiosa e lenta. Dessa forma, simuladores podem ser empregados para auxiliar na sintonia do controlador, na análise do processo e na tomada de decisão em diferentes cenários, reduzindo o tempo gasto realizando essas tarefas. Porém, o maior problema de se utilizar um simulador está no custo elevado para a obtenção de licenças, principalmente nos que simulam processos e controladores avançados. Uma solução para esse problema seria o desenvolvimento de uma ferramenta de alta qualidade com baixo custo e que tornasse possível a simulação de diferentes cenários industriais, a sintonia de controladores avançados (especificamente controladores preditivos baseados em modelo) e a análise e estudo do processo. Portanto, neste trabalho é apresentado um projeto desenvolvido junto ao PRH-34 que implementa uma interface gráfica capaz de simular processos industriais, além de controladores preditivos, na linguagem de programação Python para contornar o problema da aquisição de licenças, visto que o Python é uma linguagem aberta (*open source*) e sem custos. O software criado é apto a simular matematicamente sistemas industriais a partir de modelos matemáticos, gerando gráficos com a resposta da simulação de forma intuitiva e de fácil leitura para o usuário. Para auxiliar o uso da interface também foi implementada uma biblioteca que contém modelos de processos muito comuns na indústria e amplamente estudados na literatura.

Palavras-chave: Controle de Processos. Controle Preditivo. Desenvolvimento de Software. Python.

Abstract

The analysis and identification of industrial processes improve the engineer's perception about the actions that must be taken in several scenarios in industry. In Control Engineering, the process understanding is essential for design and configuration of controllers in order to obtain satisfactory results, however the analysis might be difficult if the process is more complex and also if advanced controllers are used, turning the design of the controller slow and costly. Therefore, simulators might be used so that it could help the configuration of the controller, analysis of the process and the decision-making in different scenarios, reducing the time spent accomplishing this task. The greatest problem in using simulators is the high cost to obtain the software's license, mainly when using advanced controllers and complex processes. A solution for this problem is to develop a high quality software with a lower cost which implemented the simulation of different industrial scenarios, design of advanced controllers (focusing on model predictive controllers) and the analysis and study of the process. Thus, this paper presents a project developed with the collaboration of PRH-34 that implements a graphic interface capable of simulating industrial processes, aside from predictive controllers, using the Python as the programming language to solve the license acquisition problem, since Python is an Open Source language. The software is capable of simulating industrial systems using mathematical models, and to plot the results. Furthermore, a tool was developed to facilitate using the interface by implementing a library that contains different process models commonly found in industry and largely studied in literature.

Keywords: Process Control. Predictive Control. Software Development. Pyhon.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

API	Application Program Interface
CARIMA	Controlled Autoregressive Integrated Moving Average
DMC	Dynamic Matrix Control
DTD	Document Type Definition
EPSAC	Extended Prediction Self-Adaptive Control
XML	Extensible Markup Language
GPC	Generalized Predictive Control
MAC	Model Algorithm Control
MIMO	Multiple-Input Multiple-Output
MPC	Model based Predictive Controller
PNMPC	Practical Nonlinear Model Predictive Control
SISO	Single-Input Single-Output
UP	Unified Process

Sumário

1 Introdução

O incentivo à pesquisa e desenvolvimento de novas tecnologias a nível mundial vem contribuindo para um avanço em diversas áreas que crescem exponencialmente nos últimos anos, como a inteligência artificial, a robótica e o processamento de sinais. Esses avanços estimularam o crescimento de simuladores de alta qualidade [1], auxiliando profissionais de diversos segmentos da sociedade a resolver problemas. A simulação é um mecanismo amplamente difundido cujo objetivo é reproduzir situações do mundo real num ambiente controlado e a ferramenta que implementa esse mecanismo é chamada de simulador [2]. Desta forma, os simuladores são fundamentais para a análise de diferentes cenários do mundo real e nesse trabalho é proposto um simulador capaz de recriar situações comuns em processos industriais, principalmente da área de petróleo e gás, utilizando modelos matemáticos para representá-los com atuação de controladores e focando principalmente em controladores preditivos baseado em modelo, ou MPC (*Model based Predictive Controller*).

Existem inúmeros tipos de algoritmo de controle utilizados na indústria, podendo se destacar na indústria petroquímica principalmente os controladores preditivos cuja área de atuação vêm crescendo com a sua pesquisa e desenvolvimento. Os controladores preditivos são ditos avançados pelo elevado grau de complexidade de sua implementação, aplicados em sistemas complexos onde é viável economicamente seu uso [3]. Dentre os diferentes algoritmos, o mais comum e utilizado neste projeto é o MPC, que é compreendido como um conjunto de técnicas que levam em consideração o conceito de predição e calculam a lei de controle adequada para o sistema utilizando algoritmos de otimização que determinam a resposta ótima através da minimização de uma função custo. Foram implementados três algoritmos MPC diferentes no projeto, o GPC (*Generalized Predictive Control*), que utiliza modelos do tipo função transferência, o PNMPC (*Practical Nonlinear Model Predictive Control*) com modelos de equações de diferença não-lineares e por fim o DMC (*Dynamic Matrix Control*) que se baseia nos modelos de resposta ao degrau. É importante salientar que também foi implementado o algoritmo de controle PID para fins de comparação de desempenho com os controladores preditivos e auxiliar na simulação e avaliação do comportamento do sistema em diferentes situações.

Desta maneira, um dos objetivos do projeto é assessorar o engenheiro de controle e automação no estudo e análise de processos industriais criando uma ferramenta que fornecesse o conhecimento básico do comportamento do processo e como este responde em diversos cenários e com controladores diferentes. A importância disso está em situações em que parar a produção é inviável, como por exemplo numa refi-

naria de petróleo cuja produção ocorre durante 24h ininterruptas, e o sistema está se comportando diferentemente do planejado, tornando possível a realização de testes em paralelo à produção para sintonizar o controlador e verificar se certas alterações resultariam em melhoras significativas durante a produção. Outra situação em que a ferramenta pode ter um papel essencial é na sintonia do controlador antes mesmo da implementação do projeto final, facilitando o design do controlador e permitindo que o engenheiro verifique as melhores possibilidades. Uma ferramenta capaz disso e que ainda possibilite a análise utilizando controladores preditivos pode ser importante para o crescimento da empresa, melhorando seus processos reduzindo custos e evitando que a produção seja interrompida para modificar parâmetros de controle ou para análise do processo.

Na simulação são utilizados os modelos matemáticos para reproduzir sistemas físicos dinâmicos. Um modelo matemático pode ser definido como uma representação abstrata da realidade por via de equações matemáticas [2], sendo uma aproximação do sistema real modelado e, desta forma, os modelos não são reproduções exatas, buscando sempre um compromisso entre a perfeição e o custo para que a melhor solução seja alcançada durante a simulação. Portanto, quanto mais fiel for um modelo, maior o esforço e tempo computacional necessário para que se tenha um bom desempenho, gerando uma dificuldade para sistemas mais complexos que requerem modelos com muitas variáveis, encarecendo o produto a ser desenvolvido e podendo até inviabilizar sua implementação. Sendo assim, o grande empecilho em se adquirir simuladores de processos industriais mais complexos se deve ao custo elevado para aquisição de licenças, devido à dificuldade no desenvolvimento de ferramentas de simulação computacional e até mesmo a ausência de interfaces intuitivas e simples para modelos complexos e com grande número de variáveis.

Portanto, outro grande objetivo deste projeto é o desenvolvimento de um software com interface gráfica capaz de simular processos industriais com controladores preditivos de forma simples e que seja implementado de forma a baratear o produto para o usuário final. Partindo dessa premissa, foram analisadas as possibilidades de implementação e concluiu-se que a melhor solução seria utilizar uma linguagem de programação *open source* que exigisse um esforço de programação reduzido e capaz de resultar num produto de alta qualidade, sendo escolhida o Python como linguagem de programação .

O Python é uma linguagem de programação de alto nível, interpretada, orientada a objetos e que tem como principal filosofia enfatizar a importância do esforço do programador frente ao esforço computacional [4], com ênfase na interatividade e privilegiando a legibilidade do código [5]. É disponível gratuitamente, possibilitando o design de uma interface sem a necessidade de obtenção de licença. Seu grande diferencial está na

disponibilidade de inúmeras bibliotecas, como de matemática simbólica, integração com sistema e *plot* interativo, criando um ambiente com infinitas possibilidades e facilitando o desenvolvimento de softwares capazes de resolver problemas complexos. Foram integrados ao Python outras ferramentas para auxiliar no desenvolvimento da interface, como por exemplo o Qt Designer, responsável pelo design da interface gráfica e o XML para armazenamento de dados.

A interface gráfica desenvolvida possui seis áreas principais, cada uma dividida numa aba separada. Cada uma dessas áreas apresenta características únicas, e são divididas igualmente dentro da interface, se complementando. Essas áreas são:

- Características do Projeto: Nesta aba são definidos os parâmetros básicos do projeto, como número de variáveis controladas, manipuladas e perturbações, além de definir um nome para o projeto e outros requisitos básicos, como nome do projeto e amostragem;
- Modelo: O usuário define as funções matemáticas dos modelos, que podem ser lineares ou não lineares. Neste caso, o usuário define funções transferências ou modelo de resposta ao degrau para modelos lineares e equações de diferença para modelos não lineares. Além disso podem ser definidos alguns parâmetros de modelos em geral, como ponto de operação e saturação do sinal de controle;
- Controlador: Os algoritmos utilizados para o controle preditivo são GPC para funções transferência, o DMC para modelo de resposta ao degrau e PNMPC para equações de diferença. O usuário define os parâmetros dos controladores, incluindo o modelo de predição. Também foi implementado um algoritmo PID e o usuário pode definir os ganhos do controlador;
- Cenário de Simulação: O cenário de simulação define como será realizada a simulação, quais as referências para a saída, o tempo de simulação e se será de malha aberta ou malha fechada;
- Simulação: Nesta aba é mostrado o resultado final, gerando gráficos com a resposta do sistema, das variáveis manipuladas e controladas;
- Índice de Desempenho: Após a simulação foi criada uma aba que mostra índices do desempenho do controlador para a análise de robustez deste pelo usuário.

Além disso, foi criada uma ferramenta com o intuito de prover ao usuário uma biblioteca com modelos tradicionais encontrados na indústria. Dessa forma, o usuário terá mais facilidade em simular processos comuns, amplamente estudados na literatura, tornando a interface mais prática.

Este projeto foi desenvolvido no laboratório do PRH-34 com orientação dos professores Daniel Martins Lima e Julio Elias Normey-Rico.

Nos Capítulos 2 e 3 será dada uma definição dos algoritmos de controle utilizados na interface, com uma breve descrição de controle preditivo e também uma descrição sobre a metodologia de desenvolvimento utilizada. Já no Capítulo 4 será detalhado passo a passo todo resultado da metodologia, desde a concepção inicial das ideias até a implementação de fato. Por fim, no Capítulo 5 é dada uma visão geral da interface, suas funções e organização, além de cada aba presente na interface detalhada profundamente.

2 Controlador Preditivo Baseado em Modelo

Neste capítulo serão introduzidas as definições básicas referentes ao *Model Predictive Control* (MPC), em português Controle Preditivo Baseado em Modelo, com uma breve explicação do funcionamento deste algoritmo de controle, dando enfâse para os algoritmos utilizados na implementação do projeto, que são: *Generalized Predictive Controller* (GPC), *Dynamix Matrix Control* (DMC) e *Practical Non-Linear Model Predictive Control* (PNMPC). A referência utilizada para o desenvolvimentos do algoritmos dos controlares preditivos foram retirados da tese de doutorado do orientador Daniel Martins Lima [6].

A estrutura básica mostrada na Figura 1 é usada para implementar essa estratégia. Resumindo, dado um *Processo* é utilizado um *Modelo* para predizer o comportamento futuro desse processo e baseando-se nesse comportamento é calculada a melhor *ação de controle* a ser aplicada. Essa melhor ação de controle é definida por uma função custo utilizada para otimizar sua resposta, cujos parâmetros são definidos pelo usuário (considerando os erros de seguimento de *referência* futuros), levando também em conta as *restrições* impostas ao modelo.

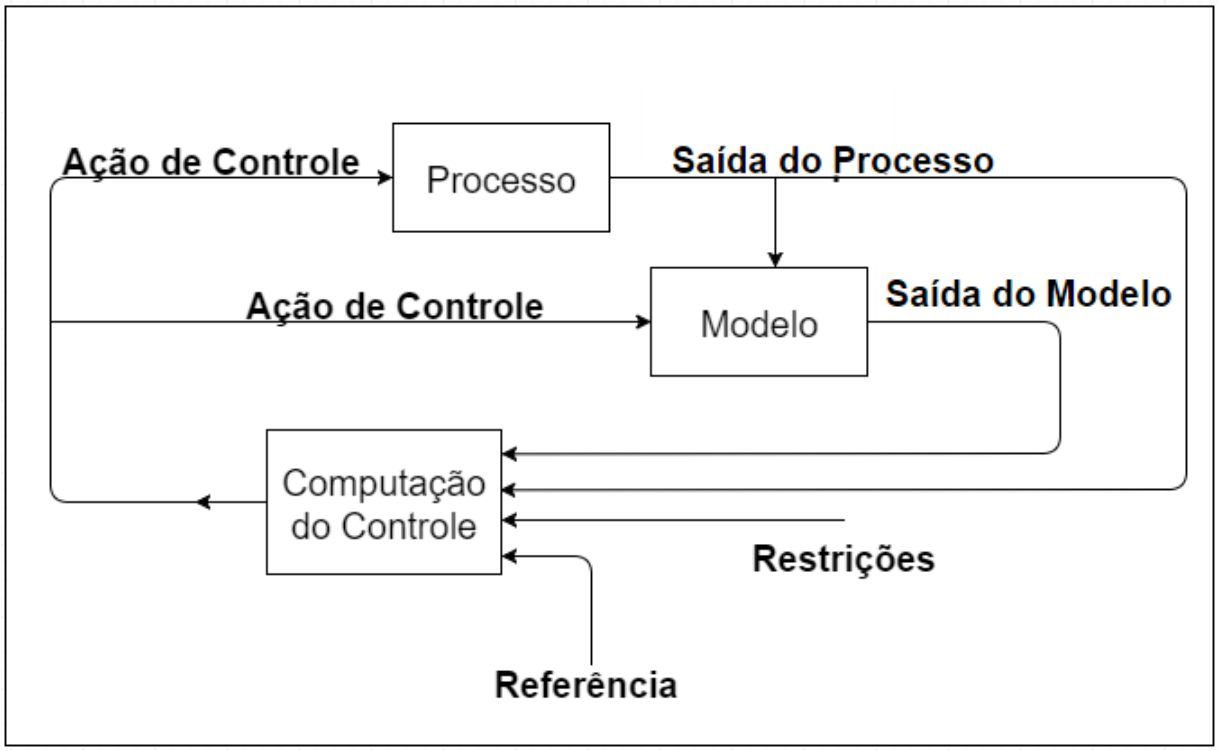

Figura 1 – Estrutura básica do MPC.

2.1 Visão Geral do MPC

Quando se utiliza o termo controle baseado em modelo não se refere especificamente à uma estratégia de controle, mas a um conjunto de métodos que tem características semelhantes, caracterizados principalmente pelo uso de modelos do processo que são utilizados para predizer o comportamento futuro do processo, e no uso de uma função custo para se calcular a ação de controle a ser aplicada no momento atual. As suas principais características são [7]:

- Uso explícito de modelos para predizer saídas do processo em instantes de tempo futuros (horizontes);
- Cálculo de uma sequência de controles minimizando uma função objetivo;
- Estratégia de horizonte deslizante, de maneira que em cada instante o horizonte é deslocado para o futuro, o que envolve a aplicação do primeiro sinal de controle da sequência calculado em cada passo.

As principais diferenças entre as variedades de algoritmos MPC são o tipo do modelo que representará o processo, as perturbações, o formato da função custo que deverá ser minimizada. Diversas aplicações de controle preditivo foram relatadas na literatura, não somente na indústria de processos, mas também em aplicações específicas de pesquisas [8–12]. Essas aplicações mostram a capacidade do MPC em alcançar sistemas de controle altamente eficientes capazes de operar sob grandes períodos de tempo com pouca intervenção.

Pode-se destacar algumas vantagens que o MPC apresenta com relação a outros métodos de controle de processos [13]:

- É atraente para usuários com pouco conhecimento em controle, visto que seus conceitos são muito intuitivos;
- Pode ser usado para controlar uma grande variedade de processos, abrangendo desde processos com dinâmica simples até outros mais complexos, incluindo sistemas com longo tempo morto, fase não-mínima e sistemas instáveis;
- Sistemas multivariáveis são facilmente tratados;
- Ele intrinsecamente compensa atrasos por tempo morto;
- É introduzido controle feedforward naturalmente para compensar a medição de perturbações;

- A sua extensão para o tratamento de restrições é conceitualmente simples e podem ser incluídos sistematicamente durante o design do processo;

- É muito útil quando as referências futuras são conhecidas;

- É uma metodologia aberta baseada em princípios que permitem desenvolvimentos mais profundos no assunto.

Por todas essas vantagens citadas, o controle preditivo baseado em modelo é uma das técnicas mais difundidas na indústria [14].

2.1.1 Estratégia do MPC

Toda a família de controladores pertencentes ao MPC possuem passos em comum e, como pode ser observado na Figura 2, existe uma ideia básica que caracteriza os controladores dessa família. As características básicas do MPC são: [6]:

1. As saídas futuras para um horizonte definido N, chamado horizonte de predição, são preditas em cada instante de tempo t usando o modelo do processo. Essas saídas preditas $\hat{y}(t+k|t)$ para $k = 1...N$ dependem dos valores conhecidos até o instante t (entradas e saídas passadas) e também do sinal de controle futuro $u(t+k|t)$, $k = 0...N-1$, que devem ser enviadas para o sistema e calculadas;

2. O conjunto de sinais de controle futuros são calculados otimizando um determinado critério de forma a manter o processo mais próximo possível da trajetória de referência $w(t+k)$ (que pode ser o próprio *set-point* ou uma aproximação dele). Esse critério geralmente assume a forma de uma função quadrática dos erros entre o sinal de saída predito e a trajetória de referência predita. O esforço de controle é incluído na função objetivo na maioria dos casos. Uma solução explícita pode ser obtida se o critério é quadrático, o modelo é linear e não existirem restrições, caso contrário um método iterativo de otimização deve ser utilizado;

3. O sinal de controle $u(t|t)$ é enviado para o processo enquanto os sinais de controle seguintes são descartados, porque, no próximo instante de amostragem, $y(t+1)$ será conhecido, o passo 1 se repetirá com esse novo valor e todas as sequências serão atualizadas. Portanto, o $u(t+1|t+1)$ é calculado (que em princípio será diferente de $u(t+1|t)$ porque novas informações foram apresentas) usando o conceito de horizonte deslizante.

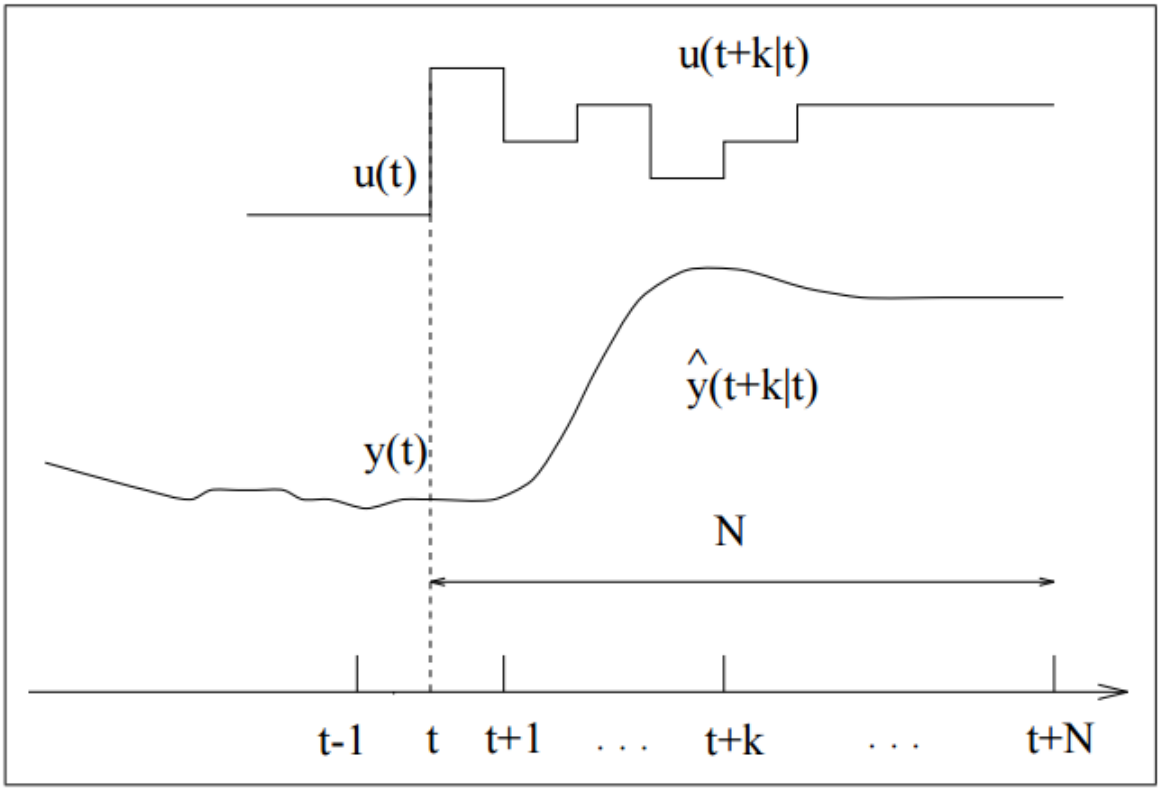

Figura 2 – Noção básica do MPC.

2.1.2 Elementos do MPC

Existem elementos em comum nos algoritmos preditivos baseados em modelo, porém a diferença é a forma como tais elementos são calculados ou até mesmo a sua estrutura, que proporciona novas opções de algoritmos MPC. Os principais elementos são [6]:

- O modelo de predição;
- Resposta livre e resposta forçada;
- A função objetivo;
- O procedimento para obter a lei de controle.

2.1.2.1 Modelo de Predição

Um dos aspectos mais importantes e que diferencia o MPC dos demais algoritmos é a sua capacidade de predizer saídas em instantes futuros $\hat{y}(t+k|t)$ e usar esta informação para calcular as ações de controle. Para que se torne possível calcular a saída predita é introduzido na malha de controle um modelo do processo. Este modelo pode ser separado em dois, o modelo do processo propriamente dito e o modelo das perturbações. Abaixo serão apresentados diferentes modelos que podem ser utilizados.

Modelo do Processo

Existem inúmeros modelos que representam o comportamento do processo e agora serão detalhados alguns dos modelos discretos mais vistos na indústria e que foram implementados na interface gráfica desenvolvida.

- Resposta ao degrau. Esse modelo considera o relacionamento entre entradas e saídas de um sistema estável dado por

$$y(t) = \sum_{i=1}^{\infty} g_i \Delta u(t-i) \tag{2.1}$$

onde g_i são os valores de saída amostrados pelo degrau de entrada $\Delta u(t) = u(t) - u(t-1)$. Embora esse modelo apresenta um somatório com infinitos elementos, posteriormente será mostrado que esse somatório pode ser truncado. Esse método é amplamente aceito na prática industrial por ser muito simples e por claramente refletir a influência de cada variável manipulada em uma saída determinada. Outra grande vantagem desse método é que caso o modelo precise ser identificado a partir de dados experimentais, não é necessário ter conhecimento a priori da estrutura do modelo, de maneira que o processo de identificação é simplificado e ao mesmo tempo permite que dinâmicas complexas, como fase não mínima e tempo morto, sejam descritas com facilidade.

- Função Transferência. Esse modelo é dado pela função transferência $G(z) = D_g(z)/N_g(z)$ de forma que a saída é

$$D_g(z)y(t) = N_g(z)u(t) \tag{2.2}$$

onde $N_g(z)$ e $D_g(z)$ são polinômios em z, variável complexa do domínio da frequência discreto. É considerado, a partir desse momento, que o processo não pode ter uma resposta instantânea, então o modelo pode ser reescrito como

$$A(z)y(t) = B(z)u(t-1) \tag{2.3}$$

sabendo que,

$$A(z) = 1 + a_1 z^{-1} + a_2 z^{-2} + ... + a_{na} z^{-na} \tag{2.4}$$

$$B(z) = b_0 + b_1 z^{-1} + b_2 z^{-2} + ... + b_{nb} z^{-nb} \tag{2.5}$$

Essa consideração é razoável na prática porque, em geral, existe um tempo morto de uma amostra entre a aplicação da ação de controle e a medição do seu efeito na saída da planta. Portanto, a predição é dada por:

$$\hat{y}(t+k|t) = \frac{B(z)}{A(z)} u(t+k-1|t) \tag{2.6}$$

A representação por função transferência é válida para sistemas instáveis e tem a vantagem de necessitar de apenas alguns parâmetros, embora a priori é fundamental o conhecimento sobre o processo no caso de identificação do modelo, especialmente a ordem dos polinômios A e B.

- Modelos Não-Lineares Genéricos. Esse modelo considera uma resposta variável

$$y_i(k+1) = f(x_i(k), u_i(k)) \tag{2.7}$$

$f(.)$ é uma função não linear discreta, com forma conhecida, das variáveis $x(k)$ e $u(k)$, com saída $y(k+1)$. Essa representação apresenta grande vantagem visto que sua definição segue funções matemáticas simples e conhecidas.

Modelo das Perturbações

Existem diversos modelos para representar perturbações, isto é, a diferença entre a saída medida e a calculada pelo modelo do processo. Na prática, modelos de degrau ou rampa são normalmente utilizados e são considerados como casos particulares de modelos de perturbações. Quando perturbações não-determinísticas são consideradas, como mudanças aleatórias ocorrendo em instantes de tempo aleatórios, o modelo ARMA (do inglês *auto-regressive and integrated moving average*) é amplamente utilizado. O ARMA é dado por:

$$\eta(t) = \frac{C(z)e(t)}{D(z)(1-z^{-1})} \tag{2.8}$$

sabendo que $e(t)$ é um ruído branco com mediana zero e os polinômios $C(z)$ e $D(z)$ são usados para descrever as características estocásticas de $\eta(t)$ [6]. Na interface foram implementados apenas perturbações simples, como do tipo degrau, rampa ou ruído branco.

2.1.2.2 Resposta Livre e Resposta Forçada

A maior parte dos algoritmos MPC se beneficiam do conceito de resposta livre e resposta forçada, uma característica explorada nos MPCs lineares. A ideia é

expressar a sequência de controle como uma soma entre diferentes sinais, como pode ser observado abaixo:

$$u(t) = u_f(t) + u_c(t) \tag{2.9}$$

$u_f(t)$ e $u_c(t)$ são relacionados, respectivamente, com a resposta livre e a forçada. O sinal $u_f(t)$ corresponde às entradas passadas e nos instantes futuros é mantido constante e igual ao último valor da variável manipulada. Isto é

$$u_f(t-k) = u(t-k), \quad k = 0, 1, 2, ... \tag{2.10}$$

$$u_f(t+k) = u(t-1), \quad k = 0, 1, 2, ... \tag{2.11}$$

O sinal $u_c(t)$ é mantido em zero nos instantes passados e igual ao próximo movimento de controle no futuro. Ou seja:

$$u_c(t-k) = 0, \quad k = 0, 1, 2, ... \tag{2.12}$$

$$u_c(t+k) = u(t+k) - u(t-1), \quad k = 0, 1, 2, ... \tag{2.13}$$

Portanto, para calcular a predição da sequência de saída, $\hat{y}(t+k|t)$ é dividido em dois: a resposta livre $(y_c(t))$, que corresponde à saída predita se for considerada que a ação de controle corresponde apenas a parte da resposta livre ,considerando o sinal de controle como $u_f(t)$. Já a segunda parte de $\hat{y}(t+k|t)$ é a resposta forçada, que considera a sequência de controle como sendo igual a $u_c(t)$. Conceitualmente, pode-se entender a resposta livre como a evolução do processo resultante do estado atual do processo e a resposta forçada seria a reação do sistema quando aplicados os controle futuros nele.

2.1.2.3 Função Objetivo

São propostas muitas funções objetivo (também conhecido como função custo) para alcançar as ações de controle e os diversos algoritmos MPC apresentam diferentes funções objetivo. Porém, existe uma proposta geral ao se utilizar uma função objetivo, que seria o seguimento de um determinado sinal de referencia (w) das saídas futuras $(\hat{y})$ num horizonte determinado, penalizando o esforço de controle (Δu) necessário para o seguimento desta referência.

No caso SISO, a função custo mais comum é representada por

$$J = \sum_{k=N_1}^{N_2} \delta(k)[\hat{y}(t+k|t) - w(t+k|t)]^2 + \sum_{k=1}^{N_u} \lambda(k)[\Delta u(t+k-1)]^2 \quad (2.14)$$

que é tradicionalmente utilizada nos algoritmos GPC e DMC.

Existem diferentes variações da função custo proposta, como por exemplo em alguns casos não se utiliza no segundo termo o incremento de controle, mas o valor do sinal de controle e, em algumas funções, o segundo termo pode ser até mesmo inexistente. A função custo apresenta alguns atributos, incluindo:

- Parâmetros: N_1 e N_2 são os horizontes de predição mínimo e máximo e N_u é o horizonte de controle, que não necessariamente é o mesmo que o horizonte de predição máximo. A interpretação de N_1 e N_2 é bem intuitiva: eles delimitam o limite do qual se deseja que a trajetória predita siga a referência. Portanto, quando um N_1 é escolhido, todos valores nos primeiros instantes até $N_1 - 1$ não são importantes. Isso acarretará numa resposta do processo suave. Em processos com um tempo morto d não é necessário que N_1 seja menor que $d + 1$ porque $u(t)$ só será influenciado a partir de $t + d$. Além disso, em processos com fase não mínima, esse parâmetro permitirá que os primeiros instantes com resposta negativa sejam eliminados da função objetivo. Os coeficientes $\delta(j)$ e $\lambda(j)$ são sequências de ponderações que consideram o comportamento futuro; geralmente valores constantes são escolhidos. Eles indicam qual variável deverá ser priorizada, ou seja, se δ é maior que λ, isso significa que a minimização da função custo resultará num erro futuro menor, enquanto, se o contrário for verdade, resultará em incrementos de controle menores.

- Trajetória da Referência: Uma das vantagens do controle preditivo é que, se a evolução futura da referência for conhecida a priori, o sistema pode reagir antes que a mudança tenha sido efetivamente realizada, evitando efeitos de atraso na resposta do processo. A evolução futura da referência $r(t + k)$ é conhecida anteriormente em muitas aplicações, como robôs, servos ou processos *batch*; em outras aplicações uma melhoria evidente na performance pode ser observada, mesmo que a referência seja constante, simplesmente sabendo o instante em que ela altera seu valor e se adiantando nessa circunstância. Na minimização da equação 2.14, a maioria dos métodos geralmente usa a trajetória referência $w(t+k)$, que não necessariamente coincide com a referência verdadeira, podendo incluir um filtro de referência.

- Restrições: Na prática, todos os processos estão sujeitos a restrições. Os atuadores tem um campo de ação limitado assim como *slew rate* (taxa de variação) determinado, como no caso das válvulas. Por razões construtivas, de segurança ou ambientais, ou até mesmo por causa do próprio escopo dos sensores, podem existir limitações nas variáveis do processo, como o níveis de tanques, vazões em tubos ou valores máximo de temperatura e pressão; ademais, as condições de operação são definidas pela intersecção de certas restrições por motivos econômicos, de forma que o sistema de controle opere dentro de fronteiras. Tudo isso requer a introdução de restrições na função que será minimizada. Muito algoritmos preditivos intrinsecamente levam em consideração as restrições normalmente definidas como fronteiras na amplitude e no *slew rate* do sinal de controle e limites da saída:

$$u_{min} \leq u(t) \leq u_{max}, \quad \forall t > 0 \tag{2.15}$$

$$\Delta u_{min} \leq u(t) - u(t-1) \leq \Delta u_{max}, \quad \forall t > 0 \tag{2.16}$$

$$y_{min} \leq y(t) \leq y_{max}, \quad \forall t > 0 \tag{2.17}$$

  Se as restrições são adicionadas na função objetivo, a minimização se torna mais complexa e uma solução linear não pode ser obtida, já que com restrições não é possível minimizar a função custo de forma algébrica. Assim, a minimização é feita de forma numérica utilizando um dos vários métodos existentes, tornando o problema mais difícil de resolver computacionalmente.

2.1.2.4 Obtendo a Lei de Controle

Para obter os valores de $u(t+k|t)$, é necessário minimizar a função J da Equação 2.14. Para implementar isso, os valores da saída predita $\hat{y}(t+k|t)$ são calculados como função dos valores passados de entrada, e da saída e dos sinais de controle futuros, utilizando o modelo escolhido e substituindo na função custo, conduzindo a uma expressão cuja minimização resulta no valores desejados. Uma solução analítica pode ser obtida pelo critério quadrático se o modelo é linear e não possui restrições, caso contrário um método iterativo de otimização deve ser utilizado para encontrar a solução. Independentemente do método, obter a solução não é fácil porque pode apresentar muitas variáveis independentes, valor que pode chegar na ordem 10 a 30 por variável manipulada. Para reduzir os graus de liberdade uma certa estrutura pode ser imposta à lei de controle. Isto pode ser obtido, por exemplo, utilizando o conceito de horizonte de

controle N_u, que consiste em considerar que após um intervalo $N_u < N_2$ não existirá nenhuma variação no sinal de controle proposto, isto é:

$$\Delta u(t+k-1) = 0, \quad k > N_u \tag{2.18}$$

que significa considerar que a ponderação de controle tem valores infinitos a partir de N_u [6].

Existem muitos algoritmos que implementam as ideias propostas anteriormente, que podem ser divididos em três grupos. O primeiro grupo é caracterizado por ter sido criado na própria indústria como uma ideia para resolver os problemas encontrados no controle dos processos. Os principais são o DMC (*Dynamic Matrix Controller*) [15] e o MAC (*Model Algorithm Control*) [12] e neles são utilizados modelos baseados na resposta ao degrau e ao impulso, respectivamente, da planta para o cálculo da predição, além das perturbações serem consideradas a diferença entre a saída real e a predita [16]. Já o segundo grupo de MPC são derivados dos conceitos de controle adaptativo [17], incluindo o GPC (*Generalized Predictive Controller*) [18] e o EPSAC (*Extended Prediction Self-Adaptive Control*) [19], utilizando modelos CARIMA para representação dos modelos. O terceiro e último grupo é utilizado para representar sistemas não-lineares, do qual encontramos o PNMPC (Practical Non-Linear Model Predictive Controller) [20].

No projeto foram implementados três tipos de controladores baseados em modelos, um de cada grupo citado anteriormente. Os algoritmos são o GPC, o PNMPC e o DMC, que serão aprofundados com mais detalhes nas seções seguintes.

2.2 Generalized Predictive Controller

O algoritmo GPC ampliado para sistemas multivariáveis (MIMO) calcula a sequência de controle segundo um critério de otimização dado por uma função custo. Para um sistema com *m* entradas e *n* saídas, a equação correspondente para a função custo é:

$$J = \sum_{i=1}^{n} \sum_{k=N_{1i}}^{N_{2i}} \delta_i(k)[\hat{y}_i(t+k|t) - w_i(t+k|t)]^2 + \sum_{i=1}^{m} \sum_{k=1}^{N_{ui}} \lambda_i(k)[\Delta u_i(t+k-1)]^2 \tag{2.19}$$

dos quais $\hat{y}_i(t+k|t)$ é a predição ótima da i-ésima saída do sistema no instante de tempo $t+k$. N_{1i} e N_{2i} são, respectivamente, os horizontes mínimos e máximos de predição para a saída i, N_{ui} é o horizonte de controle de entrada i, $\lambda_i(k)$ e $\delta_i(k)$ são as ponderações

da i-ésima entrada e saída, respectivamente, k instantes no futuro e $w_i(t+k)$ é a trajetória de referência futura para a i-ésima saída no instante $t+k$. Os horizontes totais de cada saída podem ser obtidos através da equação $N_i = N_{2i} - N_{1i} + 1$. A função custo dada é uma generalização da função utilizada para sistemas monovariáveis (SISO).

Em geral, os horizontes de entrada e saída são escolhidos de forma independente devido ao fato de que cada variável controlada apresenta atrasos e dinâmicas distintas quando relacionadas a uma variável manipulada. Também é importante ressaltar que os modelos devem ser normalizados para que as ponderações das variáveis sejam adequadas, pois, caso contrário os erros de predição e as sequências de controle não serão compatíveis na função custo e por isso a escolha das ponderações $\lambda_i(k)$ e $\delta_i(k)$ será mais difícil [13]. Uma das formas de normalizar seria dividir cada variável pelo seu valor máximo desejado [21], fazendo com que todas as variáveis tenham valor absoluto menor que um.

São utilizadas nos algoritmos GPC as equações diofantinas, que são equações cujas variáveis incógnitas a serem determinadas são na verdade polinômios, com o formato:

$$S(z^{-1})A(z^{-1}) + T(z^{-1})B(z^{-1}) = C(z^{-1}) \tag{2.20}$$

Particurlarmente nos algoritmos preditivos, são utilizados polinômios em função do operador de atraso discreto z^{-1}, como por exemplo $A(z^{-1}) = a_0 + a_1 z^{-1} + \cdots + a_n z^{-n}$. O objetivo das equações diofantinas é separar o polinômio original $A(z^{-1})$ em em dois novos polinômios $S(z^{-1})$ e $T(z^{-1})$ de forma que o cálculo da predição seja realizada de forma dependente entre as entradas futuras e as passadas, como será visto posteriormente.

O modelo utilizado para representar o sistema MIMO é o CARIMA, que é descrito de acordo com o formato DMF (Descrição Matricial Fracionária), dado por

$$A(z^{-1})y(t) = L(z^{-1})B(z^{-1})u(t-1) + D(z^{-1})v(t) + T(z^{-1})\frac{e(t)}{\Delta} \tag{2.21}$$

onde estão presentes o vetor de saídas $y(t) = [y_1(t) \dots y_n(t)]^T$, o vetor de entradas $u(t) = [u_1(t) \dots u_m(t)]^T$, o vetor de perturbações $v(t) = [v_1(t) \dots v(t)]^T$, o vetor de ruídos brancos $e(t) = [e_1(t) \dots e_n(t)]^T$ e $\Delta = 1 - z^{-1}$ que representa o operador diferença. A e L são matrizes diagonais $n \times n$ sendo que a primeira representa os denominadores das funções transferência entre entradas e saídas, e a segunda representa o atraso mínimo com relação às entradas para cada saída. B, de ordem $n \times m$, e D, de ordem $n \times p$ representam os numeradores das funções de transferência em relação às entradas e às perturbações, respectivamente. É importante destacar que os atrasos existentes

com relação às perturbações ficam implícitos em D. T é uma matriz diagonal $n \times n$ e $T(z^{-1})$ que pode ser utilizada para modelar características do ruído $e(t)$, ou como parâmetro de ajuste para melhorar a robustez do sistema em malha fechada [3].

Como a matriz $A(z^{-1})$ é diagonal, é possível obter as predições ótimas de cada saída utilizando equações diofantinas independentes, assim, o seguinte modelo MIMO é utilizado:

$$A_i(z^{-1})y(t) = z^{-di}B_i(z^{-1})u(t-1) + D_i(z^{-1})v(t) + T_i(z^{-1})\frac{e_i(t)}{\Delta} \quad (2.22)$$

onde $B_i = [B_{i1}, ..., B_{im}]$ e $D_i = [D_{i1}, ..., D_{ip}]$

Como descrito por Lima [3], com uma única diferença no aumento no número de equações diofantinas necessárias devido à quantidade superior de entradas e perturbações, obtém-se a predição ótima da i-ésima saída:

$$\begin{aligned}\hat{y}_i(t+l|t) = {} & z^{-di}H_{il}(z^{-1})\Delta u(t-1+l) + I_{il}(z^{-1})\frac{\Delta u(t-1)}{T_i(z^{-1})} \\ & + H_{v_{il}}(z^{-1})\Delta v(t+l) + I_{v_{il}}(z^{-1})\frac{\Delta v(t-1)}{T_i(z^{-1})} \\ & + \frac{H_{il}(z^{-1})}{T_i(z^{-1})}y_i(t)\end{aligned} \quad (2.23)$$

onde $H_{il}(z^{-1})$, $I_{il}(z^{-1})$, $H_{v_{il}}(z^{-1})$ e $I_{v_{il}}(z^{-1})$ são vetores polinomiais e os referentes às entradas possuem ordem $1 \times m$, e os referentes às saídas $1 \times p$. Estes vetores têm como elementos os polinômios resultantes das soluções das equações diofantinas para o i-ésima saída.

Repetindo este procedimento para as outras saídas, pode-se obter o vetor de predições futuras ótimas $\hat{y}(t)$ através de

$$\begin{aligned}\hat{y}(t) = H\Delta u(t) + H_v(z^{-1})\Delta v(t+1) + I(z^{-1})\Delta u_f(t-1) + I_v(z^{-1})\Delta v_f(t) \\ + F(z^{-1})y_f(t)\end{aligned} \quad (2.24)$$

Da mesma forma que no caso SISO, a Equação 2.24 pode ser simplificada agrupando as parcelas que não dependem das variações futuras da ação de controle na resposta livre f, resultando e,

$$\hat{y}(t) = Hu(t) + f \quad (2.25)$$

A função custo da Equação 2.19 pode ser reescrita da seguinte forma:

$$J = (Hu + f - w)^T Q_y (Hu + f - w) + u^T Q_u u \tag{2.26}$$

onde $Q_y = diag(\delta_1, \ldots, \delta_n)$ é uma matriz diagonal quadrada de ordem $\sum_{i=1}^{n} N_i$ que representa os pesos dos erros futuros e os pesos das ações de controle futuras são dadas por $Q_{\mathrm{u}} = diag(\lambda_1, \ldots, \lambda_n)$, que também é diagonal quadrada, de ordem $\sum_{i=1}^{m} N_{ui}$, onde *diag* é uma função para representar matrizes diagonais.

Rearranjando a Equação 2.26, é obtido:

$$J = \frac{1}{2} u^T P u + q^T u + f_0 \tag{2.27}$$

onde:

$$P = 2(H^T Q_y H + Q_u) \tag{2.28}$$

$$q^T = 2(f - w)^T Q_y H \tag{2.29}$$

$$f_0 = (f - w)^T Q_y (f - w) \tag{2.30}$$

O cálculo da lei de controle ocorre da seguinte maneira:

$$\Delta u = K_1 (w - f) \tag{2.31}$$

$$u(t) = u(t-1) + \Delta u(t) \tag{2.32}$$

onde K_1 é a primeira linha que faz referência a cada variável manipulada da matriz $K = (H^T Q_y H + Q_u)^{-1} H^T Q_y$ de ordem $1 \times \sum_{i=1}^{n} N_i$.

2.2.1 Algoritmo GPC

Dado um processo MIMO, com *m* entradas, *n* saídas e *p* perturbações, representado por:

$$Y(z^{-1}) = \Gamma_u(z^{-1}) U(z^{-1}) + \Gamma_v(z^{-1}) V(z^{-1}) \tag{2.33}$$

Γ_u e Γ_v são matrizes de transferência discretas, o algoritmo MIMO-GPC é executado da seguinte forma:

1. Calcular a representação em DMF do processo, obtendo as matrizes polinomiais $A(z^{-1})$, $B(z^{-1})$, $D(z^{-1})$ e $L(z^{-1})$;

2. Solucionar o conjunto de equações diofantinas e, considerando os horizontes de predição e de controle, obter as matrizes H, H_v, I_{ij}, $I_{v_{ik}}$ e F_i dado que $i = 1, \ldots, n$; $j = 1, \ldots, m$ e $k = 1, \ldots, p$;

3. Leitura das saídas e perturbações do processo e, caso $T_i(z^{-1}) = 1$ para algum i, filtrar as variáveis de perturbação, entrada e saída pelo respectivo polinômio $T_i(z^{-1})$;

4. Cálculo da resposta livre por partes:

   (a) Calcular a resposta livre da i-ésima saída através da equação;

$$\begin{aligned} f_i = & \sum_{i=1}^{m} I_{ij}[\Delta u_{fj}(t-1), ..., \Delta u_{fj}(t - d_{ij} - nb_{ij})]^T \\ & + \sum_{i=1}^{p} I_{v_{ij}}[\Delta v_{fj}(t-1), ..., \Delta v_{fj}(t - d_{v_{ij}} - nb_{ij}) + 1]^T \\ & + F_i[y_{fi}, ..., y_{fi}(t - na_i)]^T \end{aligned} \tag{2.34}$$

   (b) Calcular a resposta livre total através da abaixo e adicionar o termo $H_v \Delta v(t+1)$ caso os valores das perturbações futuras sejam conhecidos.

$$f = \begin{bmatrix} f_1 \\ \vdots \\ f_n \end{bmatrix} \tag{2.35}$$

5. Minimização da função custo:

   (a) Obter as matrizes P, q^T e f_0;

   (b) Minimizar a função custo com o uso de um algoritmo de otimização quadrática e, assim, obter o vetor de incrementos das ações de controle futuras **u**.

6. Calcular a ação de controle a ser aplicada no instante atual dado que $u_j(t) = u_{\text{j}}(t-1) + \Delta u_j(t)$;

7. Aplicar a ação de controle, e esperar um período de amostragem;

8. Voltar ao Passo 3.

Observe que o algoritmo anterior também se aplica a um processo SISO, que é um caso MIMO particular onde o número de entradas e saídas é igual a 1.

2.3 Dynamic Matrix Control

A estratégia DMC usa a função custo introduzida anteriormente na Equação 2.14, com passos semelhantes aos das estratégias introduzidas anteriormente. As diferenças estão nos cálculos das respostas forçada e livre, além das saídas preditas da planta, que são computadas utilizando modelos de resposta ao degrau [6]:

$$\hat{y}(t+k|t) = \sum_{i=1}^{\infty} g_i \Delta u(t+k-i) + \eta(t+k|t) \tag{2.36}$$

A predição da perturbação $\eta(t+k|t)$ é considerada constante durante todo o horizonte e igual a diferença entre as saídas do modelo e processo.

$$\eta(t+k|t) = \eta(t|t) = y(t) - \hat{y}(t|t) \tag{2.37}$$

Usando estas duas expressões e separando a ação de controle futuros resulta em:

$$\begin{aligned} \hat{y}(t+k|t) &= \sum_{i=1}^{k} g_i \Delta u(t+k-i) + \sum_{i=k+1}^{\infty} g_i \Delta u(t+k-i) \\ &\quad + y(t) - \sum_{i=1}^{\infty} g_i \Delta u(t-i) \\ &= \sum_{i=1}^{k} g_i \Delta u(t+k-i) + f(t+k) \end{aligned} \tag{2.38}$$

onde $f(t+k)$ é a resposta livre do sistema, isto é, a parte da resposta que não depende das ações de controle futuras, e é dada por

$$f(t+k) = y(t) + \sum_{i=1}^{\infty} (g_{k+i} - g_i) \Delta u(t-i) \tag{2.39}$$

Se o processo é assintoticamente estável, os coeficientes g_i da resposta ao degrau tendem a ser constantes após M períodos de amostragem, então $g_{k+i} - g_i$

tende a zero e pode ser considerado que:

$$\hat{y}(t+k|t) = \sum_{i=1}^{k} g_i \Delta u(t+k-i) + y(t) + \sum_{i=1}^{M} (g_{k+i} - g_i)\Delta u(t-i) \quad (2.40)$$

Note que se o processo não for assintoticamente estável, então M não existe e $f(t+k)$ não pode ser computado. Usando um horizonte de predição e um horizonte de controle, a minimização de J pode ser alcançada utilizando as predições.

2.3.1 Computando as Predições

Para computar as predições basta escrevê-las na forma matricial:

$$\hat{\boldsymbol{y}} = \boldsymbol{G}\Delta\boldsymbol{u}(t) + \boldsymbol{H}\Delta\boldsymbol{u}(t-1) + \mathbf{1}y(t) \quad (2.41)$$

onde:

$$\hat{\boldsymbol{y}} = [\hat{y}(t+1|t), \ldots, \hat{y}(t+N|t)]^T \quad (2.42)$$

$$\Delta\boldsymbol{u}(t) = [\Delta u(t), \Delta u(t+1|t), \ldots, \Delta u(t+N_u-1|t)]^T \quad (2.43)$$

$$\Delta\boldsymbol{u}(t-1) = [\Delta u(t-1), \Delta u(t-2|t), \ldots, \Delta u(t-M|t)]^T \quad (2.44)$$

sabendo que **1** é uma matriz $N \times 1$ cujos elementos são todos um. As matrizes G e H tem dimensão $N \times N_u$ e $N \times M$, respectivamente e são dadas por:

$$\boldsymbol{G} = \begin{bmatrix} g_1 & 0 & \ldots & 0 \\ g_2 & g_1 & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ g_{N_u} & g_{N_u-1} & \ldots & g_1 \\ \vdots & \vdots & \ldots & \vdots \\ g_N & g_{N-1} & \ldots & g_{N-N_u+1} \end{bmatrix} \quad (2.45)$$

$$\boldsymbol{H} = \begin{bmatrix} (g_2 - g_1) & (g_3 - g_2) & \ldots & (g_{M+1} - g_M) \\ (g_3 - g_1) & (g_4 - g_2) & \ldots & (g_{M+2} - g_M) \\ \vdots & \vdots & \ddots & \vdots \\ (g_{N+1} - g_1) & (g_{N+2} - g_2) & \ldots & (g_{M+N} - g_M) \end{bmatrix} \quad (2.46)$$

Como pode-se observar na Equação 2.41, basta calcular a matriz $\boldsymbol{G}$ para encontrar a resposta forçada $\boldsymbol{G}\Delta u(t)$. Já a resposta livre do sistema é dada por $\boldsymbol{f} = H\Delta u(t-1) + 1y(t)$ e pode ser obtida recursivamente.

2.3.2 Obtendo Resposta Livre Recursivamente

Para efeitos de simplificação, será analisado o caso SISO para descrever a recursividade do DMC. As predições em malha aberta, y_o, num instante $t+k$, sabendo informações dos instantes t e $t-1$, são [22]:

$$\begin{aligned} y_o(t+k|t) &= \sum_{i=k+1}^{\infty} g_i \Delta u(t+k-i) \\ y_o(t+k|t-1) &= \sum_{i=k+2}^{\infty} g_i \Delta u(t+k-i) \end{aligned} \tag{2.47}$$

A diferença das predições em $t+k$ nos instantes t e $t-1$ é somente o novo incremento de controle $\Delta u(t-1)$, que não era conhecido em $t-1$. Subtraindo as duas equações anteriores, as predições podem ser reescritas recursivamente como:

$$y_o(t+k|t) = g_{k+1}\Delta u(t-1) + y_o(t+k|t-1) \tag{2.48}$$

Dessa forma, para calcular a resposta livre inicialmente deve-se criar um vetor, com M elementos, e armazená-lo na memória $Y_o = [y_o(t|t-1), \ldots, y_o(t+M-1|t-1)]^T$, cujos elementos são as predições futuras dadas as ações de controle passadas conhecidas até o instante $t-1$. Durante a inicialização no instante t_0, pode ser considerado que o sistema está em regime permanente e todas as predições futuras são iguais à saída atual do sistema $y(t_0)$.

Depois da inicialização, no instante t, é necessário atualizar o vetor já que as ações de controle passadas se tornam conhecidas.

$$Y_0 = Y_0 + \begin{bmatrix} g_1 \\ g_2 \\ \vdots \\ g_M \end{bmatrix} \Delta u(t-1) \tag{2.49}$$

Portanto, o vetor Y_0 é atualizado. Depois da atualização, é necessário mover os valores dentro do vetor Y_0. Isso é importante porque no próximo instante, $t+1$, serão necessários as predições futuras do instante $t+1$ até $t+M$ baseados nas ações

de controle no instante t. Portanto, o primeiro elemento, $y_0(t|t)$, é removido do vetor e usado para calcular o erro de predição atual, $\eta(t|t) = y(t) - y_0(t|t)$. Por causa do deslocamento, o último valor de Y_0, que deveria ser $y_0(t+M|t)$ não é conhecido. Porém, no caso de sistemas estáveis, $y_0(t+M|t) \cong y_0(t+M-1|t)$, e, portanto, o novo vetor Y_0 será:

$$Y_0 = \begin{bmatrix} y_o(t+1|t) \\ y_o(t+2|t) \\ \vdots \\ y_o(t+M-1|t) \\ y_o(t+M-1|t) \end{bmatrix} \tag{2.50}$$

Por fim, para calcular a resposta livre é feito, sabendo que $\mathbf{1}_N$ é uma matriz coluna com elementos igual a 1:

$$\boldsymbol{f} = \begin{bmatrix} y_o(t+1|t) \\ \vdots \\ y_o(t+N|t) \end{bmatrix} + \mathbf{1}_N(y(t) - y_o(t|t)) \tag{2.51}$$

Os passos para o cálculo do sinal de controle são semelhantes ao do método anterior, considerando que para calcular as respostas forçada e livre são utilizados os conceitos apresentados especificamente para o algoritmo DMC.

2.4 Practical Non-Linear Model Predictive Controller

Na prática os algoritmos MPC mais utilizados são os baseados em modelos lineares. Mas, na grande maioria dos casos, os processos se comportam segundo equações não lineares, o que pode causar erros consideráveis na modelagem do processo dependendo de como o modelo foi linearizado e quais as faixas de operação definidas para o processo, dificultando a utilização de modelos lineares para representá-los.

Desta forma, foi proposto por Plucenio em [20] um algoritmo, chamado de *Practical Non-Linear Model Predictive Controller* (PNMPC), capaz de utilizar modelos não-lineares e se beneficiar da estratégia do MPC para o controle de processos. O intuito do PNMPC é, através da aplicação dos conceitos de controle preditivo baseado em modelo e modelos de sistemas não-lineares, proporcionar uma forma prática de evitar problemas de otimização não-lineares complexas. O desenvolvimento apresentado a seguir foi baseado na tese de doutorado do professor Daniel Martins Lima [6].

Como mencionado anteriormente, existem conceitos básicos que se aplicam para os algoritmos MPC. Então, sabendo que o sistema apresenta uma resposta livre e uma resposta forçada, combinando ambas resulta no seguinte vetor de predição:

$$\hat{\boldsymbol{y}} = \boldsymbol{G}\boldsymbol{\Delta}\boldsymbol{u}(t) + \boldsymbol{f} \tag{2.52}$$

Nos sistemas lineares é possível se aproveitar do conceito de superposição, representando o vetor de predição como a soma entre a resposta forçada e livre e permitindo que estes sejam calculados separadamente. Porém, em sistemas não lineares o princípio da superposição não pode ser aplicado e desta forma não é possível calcular as respostas livre e forçada em partes. Entretanto, para ultrapassar essa barreira imposta o PNMPC propõe uma solução que permite a separação das respostas linearizando a equação em cada instante de tempo.

2.4.1 Obtendo a Resposta Forçada

Para facilitar o desenvolvimento, considere um sistema SISO de primeira ordem, discreto e não-linear:

$$y(t+1) = f(y(t), u(t)) \tag{2.53}$$

onde $f : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$. Note que o valor da saída em $t+1$ depende apenas da entrada e saída em t. Essa suposição parece restritiva, mas o resultados que serão demonstrados podem ser facilmente estendidos para sistema não-lineares mais genéricos. Portanto, para maior simplicidade, esse sistema será usado.

Para prosseguir, é necessário obter o vetor de predição, mas, antes disso, é levado em consideração que $u(t) = u(t-1) + \Delta u(t)$, portanto, a predição em $t+1$ pode escrita como:

$$y(t+1) = f_0(y(t), u(t-1), \Delta u(t)) \tag{2.54}$$

As predições futuras podem ser computadas recursivamente. Por exemplo, a predição em $t+2$:

$$\begin{aligned} y(t+2) &= f(y(t+1), u(t+1)) \\ &= f(f_0(y(t), u(t-1), \Delta u(t)), u(t+1)) \end{aligned} \tag{2.55}$$

considerando que $u(t+1) = u(t-1) + \Delta u(t) + \Delta u(t+1)$,

$$y(t+2) = f_1(y(t), u(t-1), \Delta u(t), \Delta u(t)(t+1)) \tag{2.56}$$

onde $f_1(.)$ é o resultado da composição das funções. Por inspeção, é facilmente observado que o vetor de predições é:

$$\begin{bmatrix} y(t+1) \\ y(t+2) \\ \vdots \\ y(t+N) \end{bmatrix} = \begin{bmatrix} f_0(y(t), u(t-1), \Delta u(t)) \\ f_1(y(t), u(t-1), \Delta u(t), \Delta u(t+1)) \\ \vdots \\ f_{N-1}(y(t), u(t-1), \Delta u(t), \ldots, \Delta u(t+N-1)) \end{bmatrix} \tag{2.57}$$

Sabendo que N é o horizonte de predição e, para simplificar, é considerado que o horizonte de controle $N_u = N$. Essa última equação pode ser reescrita como

$$\overline{y} = \boldsymbol{F}(y(t), u(t-1), \Delta \boldsymbol{u}(t)) \tag{2.58}$$

onde $\Delta u(t) = [\Delta u(t), \Delta u(t+1), \ldots, \Delta u(t+N-1)]^T$

Utilizando a série de Taylor para aproximar a função $\overline{y}$ por uma função linearizada em cada instante de tempo, o algoritmo PNMPC computa a predição do processo não-linear da seguinte forma:

$$\overline{\boldsymbol{y}} = \boldsymbol{G_{PNMPC}} \boldsymbol{\Delta u(t)} + \boldsymbol{f} \tag{2.59}$$

cujo G_{PNMPC} é uma matriz Jacobiana de *F* com a seguinte estrutura:

$$\boldsymbol{G_{PNMPC}} = \frac{\boldsymbol{\partial F}}{\boldsymbol{\partial \Delta u(t)}} = \begin{bmatrix} \dfrac{\partial y(t+1)}{\partial \Delta u(t)} & 0 & \cdots & 0 \\ \dfrac{\partial y(t+2)}{\partial \Delta u(t)} & \dfrac{\partial y(t+2)}{\partial \Delta u(t+1)} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ \dfrac{\partial y(t+N)}{\partial \Delta u(t)} & \dfrac{\partial y(t+N)}{\partial \Delta u(t+1)} & \cdots & \dfrac{\partial y(t+N)}{\partial \Delta u(t+N-1)} \end{bmatrix} \tag{2.60}$$

Pode ser observado que a matriz $\mathbf{G_{PNMPC}}$ apresenta característica de uma matriz triangular inferior, o que é bastante intuitivo. Como o sistema é causal, as predições em $t+i$ dependem apenas dos incrementos de controle até o instante $t+i-1$.

Para computar o Jacobiano da matriz $\boldsymbol{F}$:

1. Compute e resposta livre do sistema $\hat{\boldsymbol{y}}^{\mathbf{0}} = \boldsymbol{F}(\overline{\boldsymbol{x}})$

2. Compute a primeira coluna de $\mathbf{G_{PNMPC}}$. Faça $\Delta u(t) = [\epsilon, 0, \ldots, 0]^T$, onde ϵ é um valor pequeno, como $u(t-1)/1000$ e calcule $\hat{y}^1 = F(y(t), u(t-1), \Delta u(t))$. Então, a primeira coluna de $\mathbf{G_{PNMPC}}$ é dada por:

$$\frac{\hat{y}^1 - \hat{y}^0}{\epsilon} \tag{2.61}$$

3. Compute a segunda coluna de $\mathbf{G_{PNMPC}}$. Faça $\Delta u(t) = [0, \epsilon, 0, \ldots, 0]^T$ e calcule $\hat{y}^2 = F(y(t), u(t-1), \Delta u(t))$. Portanto, a segunda coluna de $\mathbf{G_{PNMPC}}$ é dada por:

$$\frac{\hat{y}^2 - \hat{y}^0}{\epsilon} \tag{2.62}$$

4. Repita o mesmo procedimento para as outras colunas de $\mathbf{G_{PNMPC}}$ até completar a matriz.

2.4.2 Obtendo a Resposta Livre

A resposta forçada do modelo computado anteriormente é somente utilizada para encontrar a matriz $\mathbf{G_{PNMPC}}$ porque não leva em consideração a presença de perturbações e, portanto, o controle resultante não irã rejeitar esses sinais. Para computar a resposta livre corretamente o PNMPC modela as perturbações como um ruído branco integrado, o que é suficiente para descrever perturbações constantes de processos.

$$y(t+1) = f(y(t), u(t)) + \frac{e(t)}{\Delta} \tag{2.63}$$

onde e(t) é o ruído branco e $\Delta = 1 - q^{-1}$ é um polinômio no operador atraso q, isto é, $q^{-1} f(x(t), u(t)) = f(x(t+1), u(t+1))$. Multiplicando ambos lados por Δ, temos:

$$\Delta y(t+1) = \Delta f(y(t), u(t)) + e(t) \tag{2.64}$$

Como o valor esperado de $e(t) = 0, \forall t$, a predição ótima em $t+1$ é:

$$\Delta y(t+1) = \Delta f(y(t), u(t)) \tag{2.65}$$

$$y(t+1|t) - y(t|t) = f(y(t), u(t)) - f(y(t-1), u(t-1)) \tag{2.66}$$

e sabendo que a predição $y(t|t) = y(t)$, chega-se à conclusão que a saída medida é:

$$y(t+1|t) = f(y(t), u(t)) + y(t) - f(y(t-1), u(t-1)) \tag{2.67}$$

Note que $y(t) - f(y(t-1), u(t-1))$ é a predição do erro, ou seja, a diferença entre a saída em k menos a predição em $y(t|t-1)$, ou, $y(t) - y(t|t-1)$.

O mesmo procedimento pode ser usado para computar a predição em $t+2$.

$$\Delta y(t+2) = \Delta f(y(t+1), u(t+1)) \tag{2.68}$$

$$y(t+2|t) - y(t+1|t) = f(y(t+1), u(t+1)) - f(y(t), u(t)) \tag{2.69}$$

substituindo $y(t+1|t)$ da Equação 2.67 na Equação 2.69:

$$y(t+2|t) = f(y(t+1|t), u(t+1)) + y(t) - f(y(t-1), u(t-1)) \tag{2.70}$$

Por inspeção, pode ser observado que as predições podem ser calculadas recursivamente com a adição do erro de predição:

$$y(t+k|t) = f(y(t+k-1|t), u(t+k-1)) + y(t) - f(y(t-1), u(t-1)) \tag{2.71}$$

2.4.3 Algoritmo PNMPC

Portanto, para se aplicar o algoritmo PNMPC, os passos abaixo são necessários:

1. Leia a saída atual do sistema $y(t)$;
2. Compute $\mathbf{G_{PNMPC}}$, utilizando o método descrito na seção 2.3.1;
3. Calcule a resposta livre corrigida $\boldsymbol{f}$ do sistema, usando a Equação 2.71 recursivamente;
4. Minimize a função custo quadrática, reescrita da forma obtida para o algoritmo GPC:

   $$J = (\hat{y} - w)^T Q_y (\hat{y} - w) + \Delta u^T Q_u \Delta u \tag{2.72}$$

   e obtenha os incrementos de controle. A equação J é igual à utilizada no algoritmo GPC e por isso a solução é a mesma ;
5. Aplique a ação de controle calculada no processo não-linear;
6. Espere um tempo de amostra e repita.

Considerações Finais

Foi abordado nesse capítulo detalhes dos algoritmos implementados no software descrevendo a diferença entre cada estratégia MPC presente na interface desenvolvida. O DMC foi desenvolvido principalmente devido à sua grande presença na indústria petroquímica, sendo amplamente utilizado na Petrobrás e sendo alvo de muitas pesquisas e estudos. Já o GPC e o PNMPC são algoritmos com grande capacidade e que ainda terão participação cada vez maior conforme suas técnicas se tornarem mais exploradas.

3 Metodologia de Desenvolvimento de Software

Existem diversas técnicas que são empregadas para auxiliar no desenvolvimento de softwares, conhecidas como Metodologias de Desenvolvimento de Software. Pode-se considerar que uma metodologia de desenvolvimento de software é [23]:

> (...) um conjunto coerente e coordenado de métodos para atingir um objetivo, de modo que se evite, tanto quanto possível, a subjetividade na execução do trabalho. Fornecendo um roteiro, um processo dinâmico e interativo para desenvolvimento estruturado de projetos, sistemas ou software, visando à qualidade e produtividade dos projetos. (LEITE, 2006)

As metodologias são usadas com objetivos diversos, além de ter funções distintas dependendo do projeto. Uma das principais funções é a definição do dever e responsabilidade de cada agente, incluindo o usuário, o programador e o administrador. O resultado dessa divisão de trabalho é uma solidez de execução do projeto, de forma a atender as exigências básicas de implementação do projeto com clareza e permitindo uma liberdade suficiente para garantir a qualidade do produto final.

Para o desenvolvimento da interface gráfica apresentada nesta monografia foi utilizada a metodologia baseado no *Unified Process*, ou Processo Unificado, devido a sua forte conexão com a orientação a objeto, um dos paradigmas de programação da linguagem utilizada para implementação da interface. O UP possibilita a divisão do projeto em pequenas partes, subdivindo-o para facilitar a sua execução, aproveitando-se dos conceitos da engenharia do software. Cada etapa do subprojeto é executada individualmente e de forma incremental utilizando diagramas ou tabelas para simplificar a implementação.

Para melhorar a compreensão de cada etapa desenvolvida antes da implementação durante o UP, é comum integrá-lo com alguma notação que permite representar os sistemas de forma padronizada. Portanto, o Processo Unificado foi integrado com o UML (*Unified Modelling Language*), uma linguagem de estruturação e modelagem de sistemas, aplicações e comportamentos do processo, para facilitar a leitura dos diversos artefatos (diagramas e tabelas) desenvolvidos [24]. O UML apresenta uma especificação mundialmente conhecida e aceita na comunidade acadêmica, o que o torna interessante para criação dos softwares por seu reconhecimento.

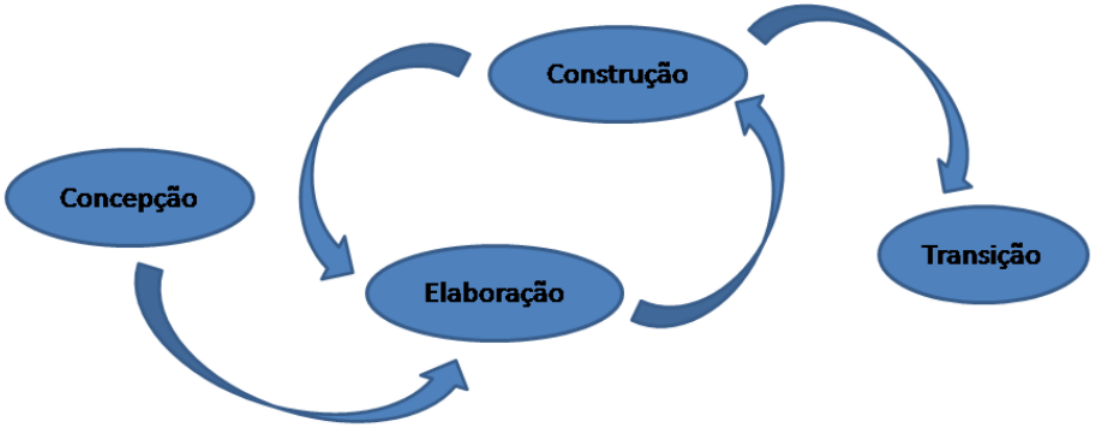

Figura 3 – Etapas do Processo Unificado.

Na Figura 3 é mostrado como as etapas são divididas e se comunicam durante a implementação do projeto.

- **Concepção**: Nesta etapa é realizada a obtenção dos requisitos de forma a verificar a viabilidade e os riscos do projeto [25], obtidos através de cenários com as interações entre clientes e usuários finais, criando a partir dos cenários os casos de uso a serem desenvolvidos.

- **Elaboração e Construção**: O projeto começa a ser planejado e modelado mais profundamente para começar a ser implementado com mais detalhes e interativamente, partindo dos requisitos de projeto definidos a priori. A implementação dos códigos é interativa de forma que ao final da interação o projeto está praticamente pronto, dependendo apenas da próxima etapa de transição.

- **Transição**: Última etapa da metodologia, no qual o projeto construído é implantado, sendo este levado até o usuário final para testes. Esta etapa não é objeto de estudo nesta monografia.

3.1 Etapa de Concepção

O principal objetivo da etapa de concepção é definir o propósito geral do objeto. Vale ressaltar que nesta fase não é necessária uma especificação detalhada dos requisitos do projeto, apenas levantamentos da estrutura básica necessária para criação e implementação, sua viabilidade e também uma estimativa do esforço e prazos necessários para realização do projeto. Inicialmente são levantados os requisitos do projeto [26].

O primeiro contato com o cliente ocorre durante a concepção, onde o analista irá descobrir quais as intenções do cliente para o projeto, produzindo um relatório com os objetivos gerais. Trata-se de uma fase com grande comunicação entre analista e cliente, visto que o analista apresenta nenhum conhecimento acerca do que será desenvolvido, sendo necessária uma análise para que ele saiba se vale a pena desenvolver o projeto requisitado.

Após comunicar-se com o cliente, um documento deve ser gerado pelo analista com uma visão geral do sistema, chamado de *Sumário Executivo*. Nele são levantados os dados relevantes, exprimindo certo requisitos básicos do projeto, assim como restrições que serão impostas nele. Como trata-se de uma visão geral, o sumário executivo pode não contemplar todas as restrições, que deverão ser analisadas e/ou implementadas em fases posteriores.

Dessa forma, os resultados da etapa de concepção geralmente envolvem o levantamento dos *requisitos do projeto*, assim como os riscos ao realizá-lo, para depois serem criados os *casos de uso* genéricos e o cronograma geral a ser seguido durante o desenvolvimento do projeto [27].

3.1.1 Requisitos do Projeto

Os requisitos são condições e capacidades que o sistema, ou projeto, deve seguir. O principal desafio em definir os requisitos está em encontrar, comunicar e lembrar o que realmente é necessário desenvolver de forma que o cliente e os membros de desenvolvimento entendam com clareza os objetivos do projeto [28].

Além disso, devem também ser definidas todas as atribuições básicas, descrevendo brevemente o comportamento do sistema para que o seu planejamento tenha clareza e facilidade durante a sua execução. Portanto, durante a fase de análise dos requisitos é necessário agrupar o máximo de informação, assim como definir as restrições do projeto, com o objetivo de representar o sistema o mais fiel possível, evitando reprojetos durante a sua realização.

Os requisitos podem ser agrupados em dois grupos, os requisitos *funcionais* que apresentam as funcionalidades do sistema e os requisitos *não-funcionais* que são as restrições ligadas a cada requisito funcional. Portanto, requisitos não-funcionais aparecem sempre conectados aos funcionais, constituindo dois tipos básicos:

- Lógicos: estão relacionados às regras de negócio da função associada;
- Tecnológicos: dizem respeito à tecnologia que será necessária para a efetuação da função.

A documentação dos requisitos é de extrema importância para o analista compreender a complexidade do sistema a ser desenvolvido. Dessa forma, é necessário um detalhamento de cada requisito que, segundo Larman (2004), sem esse detalhamento o sistema parecerá mais simples do que de fato é, explicando o porque de muitos analistas desperdiçarem muito mais tempo desenvolvendo um determinado sistema, ultrapassando os prazos e orçamentos previstos [28].

3.1.2 Casos de Uso

Uma vez que todos os requisitos tenham sido abordados, são criados grupos correlacionados, organizando-os em ciclos iterativos, chamados de *Casos de Uso*. O principal objetivo dos casos de uso é assistir no diálogo entre analistas e usuários, descrevendo a sequência lógica de como o sistema será utilizado, as interações que existem entre cada *ator*. Com isso, as funcionalidades do sistema começam a se tornar evidente, de forma que até mesmo pessoas sem nenhum conhecimento sobre o sistema são capazes de compreender suas funções e comportamentos.

Os casos de uso apresentam uma representação específica no UML, em forma de diagramas, dos quais as elipses representam casos de uso, os bonecos representam atores (usuários) e o retângulo representa a fronteira do sistema ou subsistemas, como pode-se observar na Figura 4.

É importante que nessa fase de desenvolvimento os diagramas ainda tenham alto nível de representação, pois ainda não é a intenção do analista mostrar interações detalhadas do sistema, mas sim representá-lo de forma simples e clara. Portanto, deve-se evitar um grande número de elipses e caracterizar muito bem os casos de uso para evitar que eles se tornem demasiadamente aprofundados, ou, também, que faltem funcionalidades importantes do sistema.

Com a identificação dos casos de uso, é necessário apurar se todos os requisitos funcionais do sistema estão associados de alguma forma com ao menos um dos casos de uso. Quando isso não ocorre, significa que provavelmente ainda esteja faltando algum caso de uso.

O UP é governado pelos casos de uso e centrado na arquitetura. Portanto, nas fases seguintes do desenvolvimento do sistema será detalhada progressivamente uma arquitetura que permita que cada caso de uso identificado seja devidamente desempenhado pelo seu respectivo ator. E, como dito anteriormente, o UP é interativo e incremental, o que significa que em cada etapa do desenvolvimento um conjunto de casos de uso sejam considerados para estudo e suas funcionalidades incorporadas na arquitetura [27].

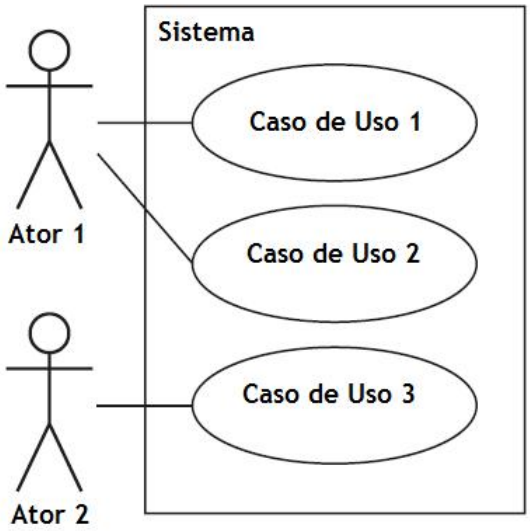

Figura 4 – Diagrama de casos de uso.

3.2 Etapa de Elaboração e Construção

A fase de construção engloba a geração do bancos de dados e a geração de código e testes. E, durante a fase de elaboração do UP, o analista começa a análise e projeto do sistema a ser desenvolvido. Nela são feitas séries de iterações onde [28]:

- A maioria dos requisitos são descobertos e estabilizados;
- A maioria dos riscos são mitigados ou até mesmo retirados;
- Os elementos principais da arquitetura são implementados e provados.

A análise realizada na elaboração do projeto geralmente é dividida em três subatividades [27]:

a) Expansão dos casos de uso;

b) Construção ou refinamento do modelo conceitual;

c) Elaboração dos contratos das operações e consultas de sistema;

É comum iniciar esta etapa da metodologia com a *expansão dos casos de uso*, visto que é uma atividade que necessita apenas dos casos de uso de alto nível definidos anteriormente e também do documento de requisitos para realizá-la imediatamente.

Já o *modelo conceitual*, que será detalhado posteriormente, depende unicamente da expansão dos casos de uso, pois o modelo conceitual utiliza informações

trocadas entre o sistema e o mundo externo que são geradas durante a expansão dos casos de uso.

Por fim, a etapa de *elaboração dos contratos* é realizada por último, já que depende diretamente do modelo conceitual e das operações e consultas do sistema [27].

3.2.1 Expansão dos Casos de Uso

A *expansão dos casos de uso* tem como objetivo a análise direta do documento de requisitos e dos casos de uso criados na etapa de concepção, especificando-os detalhadamente, seus fluxos principais e alternativos, descrevendo o comportamento natural do sistema a ser implementado e compreender todas as ações que o usuário pode realizar, os caminhos que o sistema pode seguir. Também deve-se apresentar as exceções em conjunto, definindo quais caminhos não convencionais que o usuário pode tomar no fluxo principal ou alternativo.

Durante a expansão, procura-se evitar mencionar interfaces ou tecnologias utilizadas, deve-se apenas informar o que o ator está comunicando ao sistema e, da mesma forma, o que o sistema está comunicando aos atores. Portanto, o caso de uso analisado deve ser descrito detalhadamente e deve informar as interações entre os diversos atores e o sistema.

Inicialmente são descritos todas os fluxos principais que o sistema percorre, ou seja, são esboçados os caminhos mais comuns que serão percorridos quando o sistema está funcionando da forma desejada, sem a tentativa de prever erros ou exceções. Na expansão existem alguns passos obrigatórios que devem ser cumpridos, como mostrado na Figura 5. Em todos os passos da expansão deve-se de alguma forma ficar explícito que houve troca de informações entre o sistema e um ou mais usuários.

Os passos obrigatórios em um caso de uso podem ser de dois tipos:

a) Eventos de sistema: quando alguma informação é transmitida do ator para o sistema;

b) Respostas de sistema: quando a informação é transmitida do sistema para o ator.

Depois, são identificados os fluxos alternativos, tentando observar os possíveis erros ou exceções que podem acontecer quando o sistema está percorrendo um fluxo principal, para então descrever os procedimentos necessários para resolver o problema ou tratar a exceção. É importante salientar que as exceções não necessariamente ocorrem com rara frequência, elas correspondem à eventos que, quando não tratados

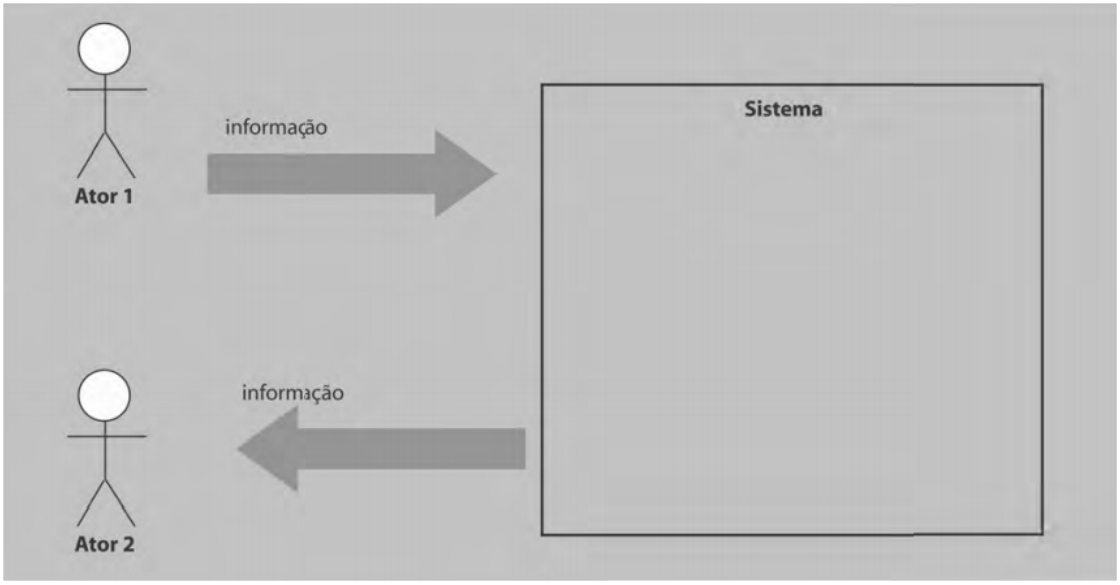

Figura 5 – Passos obrigatórios durante a expansão dos casos de uso.

como devido, podem impedir o funcionamento adequado do sistema, impedindo a conclusão do caso de uso e não que ele seja iniciado.

Na prática, as exceções apenas acontecem devido à eventos de sistema, pois quando uma informação é transmitida ao sistema ele realiza certas validações, geralmente correspondendo a restrições impostas durante a fase de concepção nos requisitos do projeto. Para cada exceção detecta-se quais possíveis ações corretivas, de forma a tratar cada exceção evitando o funcionamento incorreto do caso de uso.

Como dito anteriormente, deve-se evitar mencionar a tecnologia que será utilizada para implementação. Dessa forma, independente de como será implementado o sistema, as descrições geradas durante a expansão dos casos de uso devem conter informações que podem ser utilizadas em qualquer interface, contendo quais informações serão trocadas entre o sistema e o ambiente externo.

3.2.2 Modelagem Conceitual

O *modelo conceitual* é uma especificação do projeto que está sendo desenvolvido, independente do software de simulação que se utilizará [29]. Desta forma, o modelo conceitual visa apenas criar uma representação lógica do sistema, no mais alto nível de abstração, dando suporte para a construção de fato do sistema. O diagrama conceitual representa diferentes conceitos, cada um com seus atributos e associando esses conceitos de forma coerente.

Ele também deve descrever a informação gerenciada pelo sistema. O modelo conceitual não apresenta soluções para implementação do sistema, apenas expõe o problema. Portanto, não deve ser interpretado como a arquitetura do software (re-

presentado pelo diagrama de classes), já que esta, apesar de originar-se da modelo conceitual, já apresenta uma solução para o sistema.

Outro equívoco que muitas vezes ocorre é confundir o modelo conceitual com o modelo de dados. O modelo de dados interpreta como são representados e organizados os dados que serão armazenados no sistema, à medida que o modelo conceitual tem por objetivo representar a compreensão da informação e não a sua representação física [27]. Vendo por outro lado, um modelo de dados pode ser compreendido como uma possível representação física do modelo conceitual mais essencial.

O modelo conceitual reflete apenas o ponto de vista estático da informação. Portanto, não pode aparecer nele nada com semblante de informação dinâmica, embora na representação UML utiliza-se diagrama de classes para representar o modelo conceitual e é comum utilizá-lo para representar informações dinâmicas do sistema, como as classes. Então, o analista não deve criar modelos conceituais com métodos de classes.

Na modelagem conceitual utiliza um esquema com três tipos de elementos para representar a informação contida, utilizando diagramas de classes da UML como observado na Figura 6:

a) *Atributos*: representação que contém informações simples, como números, textos, datas e etc. Por exemplo, nome do controlador, número de entradas e tempo de amostragem;

b) *Conceitos*, representação que contém informações complexas que agregam atributos e que não podem ser descritas apenas por número e letras. Por exemplo, Controlador, Modelo, Cenário e Projeto;

c) *Associações*, que são ligações entre diferentes conceitos contendo informações específicas sobre a conexão, sendo ela mesma um tipo de informação. Por exemplo, no software deverá aparecer uma associação entre o Projeto e o Controlador, como o número de controladores que o projeto tem armazenado.

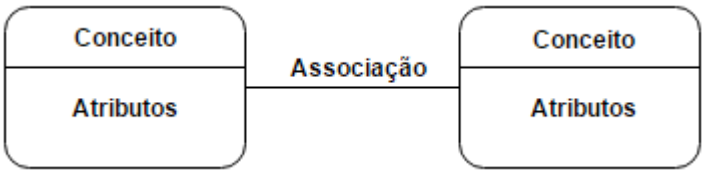

Figura 6 – Modelo conceitual representado por diagrama de classes.

Porém, a criação dos modelos conceituais apresentam complexidade maior que apenas relacionar conceitos com associações e atributos. Para que a modelagem

conceitual resulte numa representação confiável e rigorosa da informação gerenciada pelo sistema, recorre-se à diferentes práticas comuns de modelagem de sistemas. Essas práticas são:

a) *Estruturais*: representando relações de generalização estrutural de conceitos, utilizando um importante paradigma da orientação de objetos, a herança (definida nas próximas seções). Por exemplo, são implementados três tipos de controladores, que apresentam estruturas semelhantes. Assim, considera-se uma superclasse Controlador, que generaliza três subclasses ControladorPNMPC, ControladorGPC e ControladorDMC;

b) *Associativas*, representando relações de papéis associativos entre conceitos, como, por exemplo, Modelo, podendo representar junto no projeto o papel de modelo do processo ou modelo utilizado pelo controlador para predições;

c) *Temporais*, representando relações entre estados de um conceito como, por exemplo, um Cenário (cenário de simulação) e os estados simulado, não simulado.

Por fim, é muito importante garantir um modelo conceitual que resulte num banco de dados estruturado e normalizado, garantindo que as informações sejam representadas inconsistentemente e evitando dados irrelevantes. Um bom modelo conceitual simplifica a criação de códigos porque o programador não precisará verificar continuamente a consistência do sistema, que é garantida pelo modelo conceitual.

3.2.3 Contratos

As diversas operações e consultas que serão implementadas no projeto podem ser descritas utilizando contratos definidos ainda durante a etapa de elaboração do projeto. As *operações de sistema* são métodos que são acionados quando ocorre um evento de sistema, portanto, como uma reação a uma ação do usuário. Podem também ser interpretadas como fluxos de informação que fluem do exterior para o interior do sistema, alterando informações gerenciadas pelo sistema. Já as *consultas de sistema* apresentam papel inverso, as informações já armazenadas são verificadas internamente.

Então, os contratos descrevem quais as funções das operações e o que será cumprido pelas operações. São determinadas pré- e pós-condições que descrevem mudanças de estados do sistema e também exceções. As *pré-condições* definem quais as condições iniciais que o sistema deve obedecer para que a operação ou consulta seja executada. As *pós-condições* representam as mudanças que ocorreram no sistema após a execução da operação ou consulta. Por fim, as *exceções* que representam

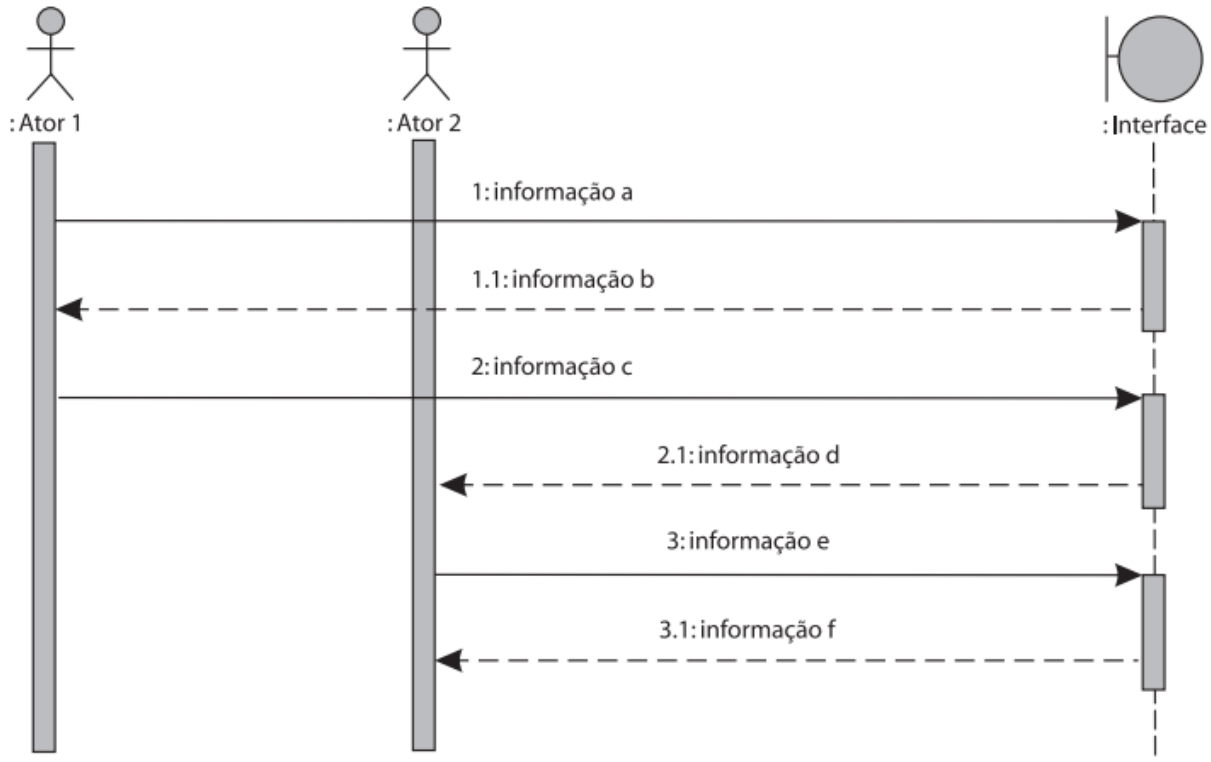

Figura 7 – Diagrama de sequência.

estados que devem ser avaliados durante a execução que pode acarretar na não execução final da operação [3].

3.2.4 Diagrama de Sequência

Os diagramas de sequência são comumente utilizados para representar uma sequência de processos realizadas pelo usuário, cumprindo com os contratos definidos para cada operação para executá-las. Seu principal objetivo é mostrar interações entre diferentes objetos a partir de casos de uso, representando uma linha de tempo no qual eventos podem ocorrer.

Outra forma de interpretar os diagramas de sequência é como uma ferramenta que sistematiza os casos de uso expandidos desenvolvidos previamente com o intuito de refiná-los, obtendo mais detalhes sobre o funcionamento do sistema.

A UML apresenta um diagrama benéfico e específico para representar a sequência de eventos do sistema numa situação de um caso de uso, como pode ser observado na Figura 7. O diagrama de sequência apresentado tem como elementos instâncias de *atores*, interpretado por figuras humanas, e instâncias que representam elementos do sistema (interface).

Além disso, para representar as mensagens trocadas são utilizadas linhas

verticais, podendo ser tracejadas ou inteiras dependendo do tipo de mensagem. As mensagens tracejadas consistem em retornos de mensagens pelo sistema, representando então a reação do sistema aos atores quando estes provocam um estímulo ao sistema. No diagrama representado pela Figura 7 existem três tipos de envio de mensagens:

a) *entre atores*, correspondendo a passos complementares do caso de uso expandido;

b) *dos atores para o sistema*, eventos de sistema do caso de uso expandido;

c) *do sistema para os atores*, respostas do sistema do caso de uso expandido.

Porém, é possível interpretar os tipos de envio de mensagens de outra forma, dividindo-os em quatro tipos:

a) *evento de sistema*: representa mensagens enviadas pelo ator para o sistema, sendo caracterizado por uma seta do ator para a interface;

b) *resposta do sistema*: acontecem quando o sistema envia alguma informação para os atores, sendo representado por uma seta tracejada da interface para o ator;

c) *operação do sistema*: quando o sistema precisa alterar alguma informação interna, ele faz uma operação de sistema. Portanto, é uma chamada de método executada internamente em resposta a um evento do sistema;

d) *consulta do sistema*: são semelhantes a operação do sistema, porém nesse caso um método é invocado apenas para que o sistema retorne alguma informação para os atores, sem alterar dados contidos internamente.

Todos os elementos do diagrama de sequência, como atores ou interfaces, dispõem de uma linha de tempo representado por colunas verticais, onde eventos podem ocorrer. Se a linha do tempo permanece tracejada então o elemento que a constitui está inativo. Já quando ela está cheia, o elemento está ativo, o que pode sugerir que o elemento está operando ou está esperando o resultado de alguma operação. Atores humanos estão sempre ativos.

3.2.5 Diagrama de Colaboração e de Classes

Após todas as análises anteriores, pode-se começar a desenvolver soluções para o projeto de software propriamente dito. O problema já está descrito no diagrama conceitual, diagramas de sequência dos casos de uso e também no contratos e,

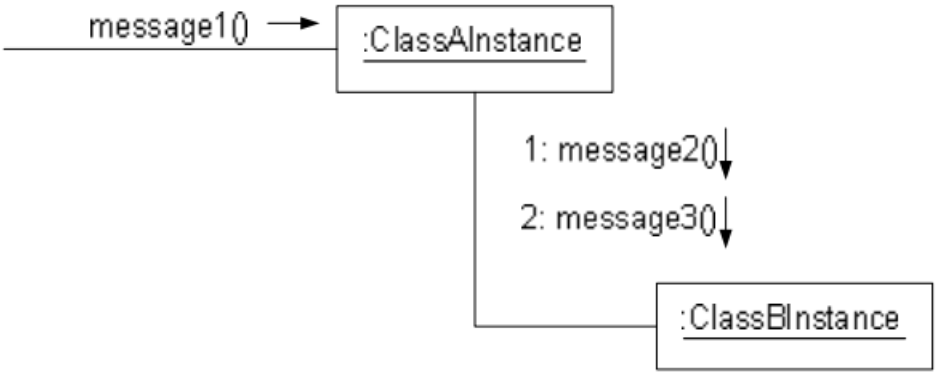

Figura 8 – Diagrama de colaboração.

portanto, apenas falta o desenvolvimento do projeto de software para finalizar o sistema [3]. Pode-se dividir o desenvolvimento do projeto em duas partes:

a) *o projeto lógico*, evolução do modelo conceitual e diagramas de sequência resultando nos diagramas de classes e colaboração, representando a interação entre os objetos quando operações e consultas de sistemas são executadas;

b) *o projeto tecnológico*, que representa a solução inerente a tecnologia, como por exemplo: interface a ser desenvolvida, como será feito o armazenamento dos dados internos, a comunicação entre os diversos atores e etc.

Dessa maneira, são definidas duas atividades, o *diagrama de colaboração* e o *diagrama de classes*. O diagrama de colaboração permite a criação de interações entre classes de objetos, mostrando como as pós condições dos contratos serão realizadas, expressando detalhadamente a comunicação entre os objetos. As mensagens enviadas definem os métodos que serão implementados nas classes do projeto, podendo ser básicas ou delegadas. As básicas acontecem quando o objeto que recebeu a mensagem simplesmente executam a operação enviada, já as delegadas são mensagens que são passadas adiante para o próximo objeto, de forma que o acoplamento entre os objetos não aumente a complexidade das classes envolvidas. Portanto, com o diagrama de colaboração pode-se determinar as interações entre as instâncias de objetos, que se comunicam via mensagens, como observado na Figura 8.

Depois de criado os diagramas de colaboração é possível criar o diagrama de classes que descreve os objetos do sistema e os relacionamentos que os diferentes objetos apresentam entre si, com seus respectivos métodos a serem inseridos entre as associações. Primeiramente, o diagrama de classes é uma cópia do de colaboração, para depois desenvolver mais profundamente os atributos e métodos das associações, além das operações atribuídas a cada classe. Também são criados atributos particulares de cada classe, que não são “enxergadas” por classes diferentes, representando a

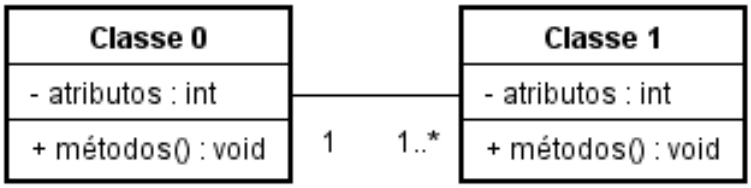

Figura 9 – Diagrama de classes.

dinâmica interna do objeto. As modificações básicas que devem ser feitas no diagrama conceitual para a criação do diagrama de classes são, basicamente [27]:

a) *adição dos métodos*;

b) *possível detalhamento dos atributos e associações.*;

c) *possível alteração na estrutura das classes e associações.*;

d) *possível criação de atributos privados ou protegidos*;

e) *adição da direção das associações.*.

Portanto, o diagrama de classes apresenta mais informações com relação ao modelo conceitual, acrescentando mais *métodos* e *associações* ao diagrama, aumentando a complexidade utilizando o conceito de *herança*. A herança consiste numa técnica que possibilita que classes com atributos e métodos semelhantes possam ser agrupadas numa única classe que terá essas características inerentes [30]. Portanto, a herança permite que sistemas altamente complexos sejam simplificados com reaproveitamento de código, generalizando associações entre classes [3].

Como pode ser observado na Figura 9, o diagrama de classes possui elementos básicos para sua representação UML:

- *Atributos*, que determinam as características básicas do objeto. Essas características podem incluir, por exemplo, nome da classe, tipo de dado do atributo, valores iniciais;
- *Métodos*, definem as funções que os objetos, que podem ter nomes e receber parâmetros para execução da função;
- *Associações*, exprimem a relação entre cada classe descrita no diagrama, com intuito de associar e criar uma rede de comunicação entre classes.

Quando terminados os detalhamentos do diagrama conceitual que resultam no diagrama de classes do projeto, termina-se o desenvolvimento do projeto lógico, o que

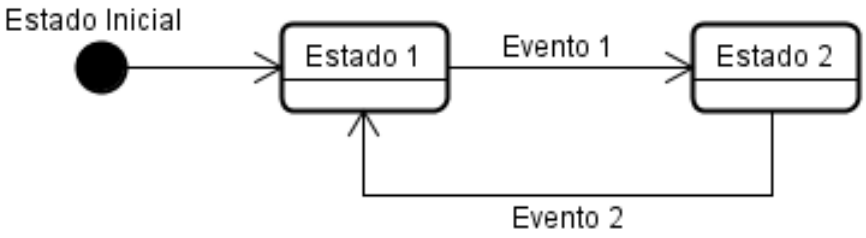

Figura 10 – Diagrama de estados.

significa que existem informações suficiente para implementar as classes e representar toda a lógica de transformação e apresentação dos dados do sistema. Porém, ainda pode-se utilizar mais um diagrama comum ao UML para facilitar a compreensão do ciclo de vida do sistema e os caminhos percorridos por ele, chamado de diagrama de estados, que permite fazê-lo em paralelo a criação de códigos e lógicas de programação.

3.2.6 Diagrama de Estados

A última etapa da metodologia consiste na criação do diagrama de estados, que modela todos os estados em que o sistema percorre durante seu ciclo de vida, sendo uma descrição abstrata do comportamento do sistema com clareza e de forma simplificada. Nessa etapa o código propriamente dito é implementado, com a criação dos códigos de cada classe correspondentes à camada de domínio da aplicação [3], ou seja, é implementada toda a lógica interna que foi definida anteriormente para cada classe. Também é feita a associação entre a camada de domínio e a de persistência, de modo que os dados preservados possam ser recuperados pelo usuário.

Os diagramas de estados mostram o ciclo de vida de um dado objeto, os *eventos* que ele experiencia, as suas *transições* e os *estados* que o objeto se encontra durante as transições. Se um evento é acionado que não está representado no diagrama de estado, este evento é ignorado completamente ou analisado para tratamento durante o desenvolvimento. Portanto, os elementos básicos do diagrama de estados são:

- *Estados*, que representam a situação ou condição que o objeto se encontra em determinado instante, dependendo diretamente dos próximos elementos. Todo objeto possui um estado inicial, que pode ou não ser retornado futuramente;
- *Eventos*, que definem situações dos quais ocorrem uma transição, podendo mudar ou não o estado do objeto;
- *Transições*, que são definidos por setas no diagrama UML, demonstrando a relação entre dois estados diferentes, modificando o estado de um objeto para outro quando um evento específico é acionado.

3.3 Camada de Persistência

É muito importante para o usuário a preservação dos dados após uma simulação evitando retrabalho e facilitando o uso da interface gráfica. Dessa forma, utiliza-se o conceito de persistência para o armazenamento eficaz [31] e automático dos dados em mídias não-voláteis, como HD do computador [3], permitindo recuperação dos dados até mesmo quando o sistema está inativo ou quando é reiniciado.

Os tipos de variáveis armazenadas internamente são todas aquelas que são suficientes para recriar a simulação exatamente igual à realizada inicialmente, ou seja, são armazenados apenas os dados necessários para o usuário ter condições de simular um projeto novamente, para evitar que sejam armazenadas muitas informações irrelevantes.

A camada de persistência foi implementada utilizando o formato XML (*Extensible Markup Language*) para salvar os dados, que é uma linguagem de marcação que descreve uma classe de objetos chamada de documentos XML, descrevendo parcialmente o comportamento de programas que os processam [32], definindo um conjunto de regras das quais esses documentos devem seguir de forma que evidencie o que está documentado tanto para o humano quanto para a máquina. Ela foi criada pela W3C, uma das principais organizações de padronizações da *World Wide Web*, que mantem todas suas especificações *open source*, permitindo o caráter aberto e a facilidade do desenvolvimento de uma interface sem a necessidade de obtenção de licenças.

3.3.1 Extensible Markup Language (XML)

Para interpretar os documentos XML geralmente são utilizados pacotes de softwares específicos, também conhecidos como XML *parsers* ou processador XML. Os elementos básicos de um documento XML, mostrados na Figura 11, são [33]:

- *Declaração*: Sempre aparecendo na primeira linha com os caracteres <?, indica a versão do XML sendo utilizado, além de algumas informações acerca da documentação do código;
- *Elementos*: Os elementos representam alguma informação, sempre iniciando com <> e finalizando com </>, possuindo uma *tag* específica (representada na figura por 'Elemento'). Percebe-se que um elemento pode ter no seu corpo outros elementos declarados com as suas próprias *tags*.
- *Atributo*: Conferem características aos elementos, assumindo valores ou *strings*, por exemplo. Um elemento pode ter quantos atributos forem necessários.

Dessa forma, são utilizadas *tags* para determinar os tipos de objetos armazenados, criando elementos de diferentes tipos e combinações. A estrutura hierárquica do XML é perfeita para a representação do conceito de programação orientada a objeto, permitindo a criação de classes complexas, com atributos específicos e implementando uma camada de persistência simples e objetiva.

Mas também existem outras vantagens ao se utilizar o XML como linguagem de marcação. Devido a sintaxe simples utilizada, o XML é uma linguagem de fácil leitura para humanos e máquinas, permitindo uma compreensão maior para os desenvolvedores e analistas, facilitando a análise do que está sendo armazenado e permitindo que alterações sejam feitas de forma simplificada. Também é considerada uma linguagem multi-plataforma, sendo interpretada em máquinas com diferentes plataformas instaladas permitindo a cambiabilidade da informação.

Outra característica importante do XML é a validação de documentos. Sempre que documentos XML são construídos e interpretados pelo *parser* é possível especificar a priori como estes documentos devem ser formatados. Portanto, todo documento XML só será utilizado se ele estiver conforme um formato padrão criado anteriormente, evitando com isso erros com arquivos corrompidos ou que perderam informações. Essa padronização é realizada num documento onde todos os tipos de elementos, seus atributos, a ordem de aparição de cada elemento e outras informações relevantes são definidas, sendo chamadas de XML *schema*.

3.3.2 XML Schema

Quando se fala em *tipagem de documentos* pode-se fazer uma analogia com os tipos de linguagem de programação. Geralmente, linguagens de programação são utilizadas para descrever estruturas que podem ser compostas de formas particulares e *tipagem de documentos* também podem ser vistos dessa forma. Os componentes primitivos e os tipos de composições possíveis de cada caso que são diferentes, porém conceitualmente são semelhantes. Uma *tipagem de documento* é, geralmente, conhecida como um *schema*.

Os *schemas* são particularmente valioso por diversas razões, mas principal-

```
<?xml version="1.0" ?>
<Elemento>
    <Elemento2>Dado</Elemento2>
    <Elemento3 atributo="valor"/>
</Elemento>
```

Figura 11 – Esquema básico do XML.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:element name="Processo">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="Tipo" type="xs:string"/>
                <xs:element name="Nome" type="xs:string"/>
                <xs:element name="Amostragem" type="xs:float"/>
                <xs:element name="Modo" type="xs:string"/>
                <xs:element name="Redes" type="listaRedes"/>
                <xs:element name="Instrumentos" type="listaInstrumentos"/>
                <xs:element name="Controladores" type="listaControladores"/>
                <xs:element name="QTEntradas" type="xs:nonNegativeInteger"/>
                <xs:element name="QTSaidas" type="xs:nonNegativeInteger"/>
                <xs:element name="QTPerturbacoes" type="xs:nonNegativeInteger"/>
                <xs:element name="NomesEntradas" type="xs:string"/>
                <xs:element name="NomesSaidas" type="xs:string"/>
                <xs:element name="NomesPertubacoes" type="xs:string"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
</xs:schema>
```

Figura 12 – Exemplo de um documento *schema*.

mente por duas: eles requerem um pensamento crítico quanto ao design da aplicação e dos dados descritos e são muito usados para ajudar a especificar como certos documentos devem ser construídos e interpretados. Dessa forma, são muito importantes para a definição da estrutura que os documentos XML, por exemplo, irão apresentar [33].

Existem diversas linguagens para a definição dos schema, sendo a mais encontrada a DTD (*Document Type Definition*), que foi a utilizada no projeto. O DTD foi escolhido já que permite uma grande exatidão e flexibilidade na definição das estruturas e dos elementos do documento XML. Na Figura 12 pode ser observado um exemplo de um schema criado para representar a estrutura de um documento XML, composto por um elemento chamado *Processo*, que nele são atribuídos diversos elementos numa sequência. Pode-se observar que existem diferentes tipos de elementos, os complexos, que são compostos por outros elementos, e os simples, que são definidos como números, textos e etc. A sintaxe utilizado é bastante similar da utilizada para definição dos documentos XML, mostrando a facilidade de se estruturar utilizando o DTD.

3.4 Linguagem de Programação - Python

Criado no fim dos anos 80, o Python surgiu numa época em que o desenvolvimento tecnológico computacional estava em crescimento exponencial, impulsionados

por empresas que estavam em crescente avanço no mundo, como Microsoft e Apple. Inspirado pela tradição de se nomear uma linguagem de programação homenageando uma pessoa famosa, Guido Van Rossum criou o Python inspirado na série britânica *Monty Python's Flying Circus* em 1989 [34].

O Python é uma linguagem de programação de larga escala e evoluiu das raízes do mundo dos *scripts*, desenvolvendo-se de forma orgânica pela visão e liderança do seu criador. Como a maioria das linguagens de *script*, o Python caracteriza-se pela excelente capacidade de manipulação de textos e arquivos. Porém, diferentemente da maioria das linguagens de *script*, o Python apresenta um ambiente de orientação a objetos poderoso com uma API (*Application Program Interface*) robusta para programação de redes, *threads* e desenvolvimento de interfaces de usuário gráficas [33].

Considera-se que o Python trouxe para o 'mundo dos *scripts*' a robustez e a sanidade, da mesma forma que o Java trouxe para o mundo do C++. Mas, assim como ao utilizar o Java no lugar do C++, houve uma simplificação da linguagem com forte suporte em orientação a objetos. Mudando para uma linguagem mais simples removeu-se os detalhes de baixo nível em gestão de memória e hardware em linguagens como C++, porém ganhou-se em robustez e facilidade em encontrar erros nos códigos. Como esperado, a flexibilidade teve um custo, já que o Python pode resultar em piores performances que outras linguagens como C e C++.

Mas o Python apresenta algumas características que o tornam atraente com relação a outras linguagem de programação [33]:

- A leitura e manutenção do código-fonte do Python é simples e fácil;
- O interpretador interativo simplifica o uso de código fragmentados;
- O Python possui uma portabilidade excelente, mas também não restringe o acesso às capacidades de outra plataformas específicas;
- As características de orientação a objetos são poderosas e acessíveis.

Dessa forma, foi escolhido o Python para criação da interface desenvolvida no projeto. Para melhorar a legibilidade e documentação do código criado foi utilizado o *Doxygen*, uma ferramenta (Figura 13) para geração de documentação de códigos, que suporta as linguagens de programação mais comuns. O *Doxygen* pode ser útil de três forma distintas [35]:

1. Pode gerar uma documentação *online*, criando um *browser* via HTML e/ou um manual de referência *offline* (em *Latex*) a partir de um arquivo fonte documentado. A documentação é extraída diretamente das fontes, o que permite manter mais facilmente uma documentação consistente com o código fonte;

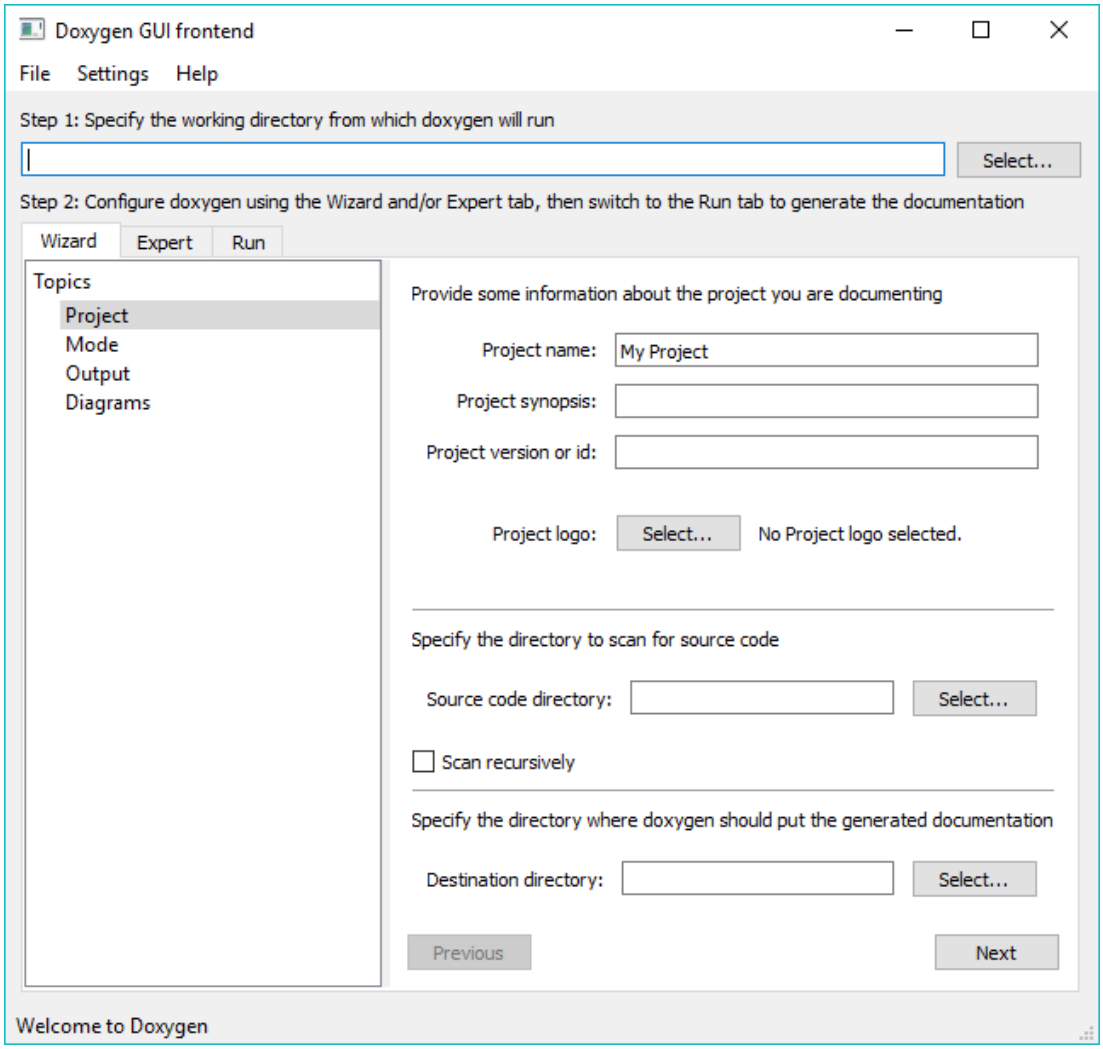

Figura 13 – Software Doxygen.

2. É possível configurar o *Doxygen* para extrair a estrutura do código de arquivos fonte não documentados. Essa função é muito útil para se encontrar rapidamente quando são utilizados fontes muito grandes;

3. O *Doxygen* também permite visualizar a relação entre os vários elementos a partir de gráficos de dependência, diagramas de herança e diagramas de colaboração, todos gerados automaticamente, facilitando a leitura da documentação do código e o seu entendimento.

Considerações Finais

Neste capítulo foi apresentada a metodologia utilizada para o desenvolvimento do projeto da interface gráfica. Durante sua implementação, o projeto foi alterado diversas vezes e sempre que possível tentou-se manter as características descritas

durante as etapas da metodologia, procurando manter a coerência no design do software. Além disso, foi descrito brevemente a linguagem de programação utilizada para desenvolver a interface, assim como as ferramentas que foram necessárias para sua elaboração, tentando destacar o porque da sua escolha mostrando as suas qualidades.

4 Requisitos da interface e Projeto de software

No capítulo anterior foi detalhado o Processo Unificado, metodologia utilizada para desenvolver o projeto. Assim, seguindo os passos descritos pela UP, o projeto é iniciado pela fase de concepção, do qual é criado um documento com uma visão geral do que deve ser desenvolvido, procurando definir suas funções e comportamentos básicos. Esse documento, chamado de *Sumário Executivo*, originou-se após a análise junto ao orientador da empresa do que deveria ser o projeto, visto que o sumário executivo visa justamente dar ao analista uma noção do projeto e as suas restrições mais básicas. O resultado é mostrado na Tabela 1:

Tabela 1 – Sumário executivo gerado na fase de concepção da UP

Sumário Executivo
Criar um software com uma interface gráfica que visa simular o controle de processos industriais utilizando modelos matemáticos de plantas reais da indústria petroquímica. Não será função do sistema identificar modelos, estes já estarão definidos pelo usuário e/ou estarão armazenados num banco de dados do sistema, gerando uma biblioteca, também fornecendo um modelo inicial capaz de realizar as simulações necessárias. Deverá ser capaz de simular modelos lineares e não-lineares assim como simular diversos modelos simultaneamente, além de sistemas monovariáveis e multivariáveis. Os sistemas serão discretos e portanto um período de amostragem deverá ser definido corretamente pelo usuário. A simulação em si do controle deve prover diversos tipos de controladores, tendo foco em controladores preditivos. O sistema deve ser capaz de avaliar a robustez dos controladores para diferentes fatores que gerem erros e avaliar o índice de desempenho. Além disso os cenários de simulação serão definidas pelo usuário, podendo este definir, por exemplo, diferentes tipos de entrada, como referência ou perturbações e entre outras condições. Também poderá ser feita simulação em malha aberta para se analisar os modelos. O simulador deverá ser capaz de prover uma janela de visualização de dados da simulação para o usuário analisar o resultado obtido de forma objetiva. Além disso, a interface deverá apresentar uma ferramenta com diversos modelos já estudados na indústria armazenados num banco de dados. A interface gráfica será implementada na linguagem de programação Python pela facilidade e o alto nível de abstração em conjunto com um software de desenvolvimento de interfaces do próprio Python, o Qt Designer. Deverá prover uma forma simples e clara para o usuário ter facilidade ao utilizar o sistema e um algoritmo de simulação eficiente.

4.1 Requisitos da Interface

Uma vez que o sumário executivo é finalizado, é criada uma lista completa com os requisitos funcionais do projeto, definindo as funções básicas do software ou parte dele, além dos requisitos não-funcionais, que estão relacionados com desempenho, usabilidade, disponibilidade e com as tecnologias envolvidas do sistema, agrupando os dois tipos de requisitos numa tabela similar à Tabela 2. Nela é observado a construção do requisito *Armazenar Modelos da Biblioteca*, que foi dada a código (F12), detalhando também os requisitos não-funcionais e as restrições impostas. As funcionalidades do sistema foram divididas em três grupos: (i) Interface Gráfica (ii) Algoritmo da Simulação (iii) Armazenamento de dados.

Tabela 2 – Descrição do requisito Armazenar Modelas da Biblioteca

F12 – Armazenar Modelos da Biblioteca
Descrição – o sistema deve ser capaz de armazenar todos os modelos dos processos desejados numa biblioteca, de forma que seja possível que o usuário utilize os modelos na interface posteriormente para simulações.
Restrições Não-Funcionais
NF1.1 – usuário não pode alterar os modelos armazenados dentro da biblioteca.
NF1.2 – armazenamento deve ser implementado utilizando XML pela facilidade de interação com Python.
NF1.3 – não armazenar todos parâmetros, apenas os suficientes para que a simulação possa ser realizada. Os únicos parâmetros armazenados são: Nome do projeto, número de variáveis de processo, perído de amostragem, lista dos modelos, controladores e cenários contendo as informações básicas de cada um.

Para cada grupo encontrado foram definidas diferentes funcionalidades, descritas a seguir, que são mostradas mais detalhadamente no Apêndice A:

(i) Interface Gráfica

- (F1) Definição dos Parâmetros do Projeto
- (F2) Definição dos Modelos
- (F3) Definição dos Controladores
- (F4) Configuração do Cenário de Simulação
- (F5) Visualização dos Gráficos de Simulação

(ii) Algoritmo da Simulação

- (F6) Calcular Ação de Controle
- (F7) Calcular Índices de Desempenho

(F8) Implementação da Simulação

(F9) Gerenciamento do Banco de Dados

(F10) Gerenciar Dados do Processo

(F11) Atualizar Dados do Processo

(iii) Armazenamento de dados

(F12) Armazenar Modelos da Biblioteca

(F13) Salvar Dados da Simulação

(F14) Abrir Dados de Simulações Armazenadas

4.2 Levantamento dos Casos de Uso

Com os requisitos devidamente coletados, pode-se levantar o diagrama de casos de uso do sistema. Como já citado anteriormente, o diagrama visa auxiliar a comunicação entre os analistas e o usuário, descrevendo a sequência lógica de como o sistema deve ser utilizado, de forma que fique evidente as funcionalidades do software em questão.

Como observado na Figura 14, foram encontrados os Casos de Uso, que representam os principais negócios realizados pelo sistema. Os Casos de Uso são: *Configurar Projeto*, *Configurar Modelo*, *Configurar Controlador*, *Configurar Cenários de Simulação*, *Calcular Lei de Controle* e *Gerar Gráficos*. Pode-se interpretar o diagrama como:

- O usuário inicialmente configura o projeto, definindo os parâmetros básicos da simulação;
- Posteriormente ele configura os modelos a serem utilizados na interface;
- Então o usuário configura os controladores, seus parâmetros e variáveis desejadas.
- Por fim o usuário define o cenário da simulação, as referências, tempo de simulação e outros parâmetros, realizando a simulação.
- O processo executa, analisa as configurações e calcula uma lei de controle adequada para o cenário escolhido pelo usuário.
- Então o processo gera os gráficos da simulação realizada conforme solicitado pelo usuário.

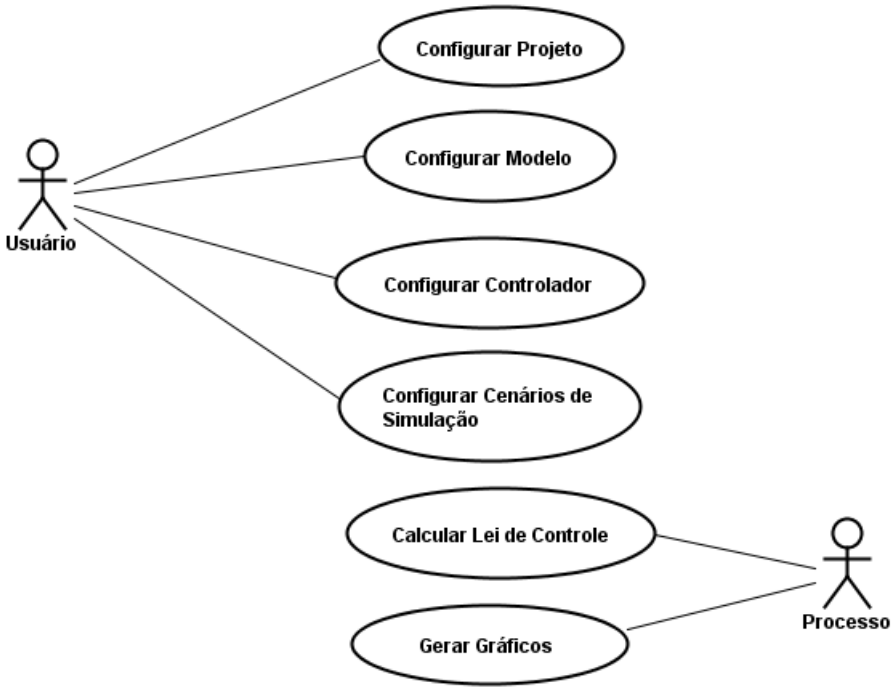

Figura 14 – Diagrama de caso de uso gerado para o sistema.

Observa-se que de maneira sucinta e simplificada pode-se descrever todo o processo de simulação, as funcionalidades do sistema e o seu comportamento perante as ações do usuário e assim, o analista apresenta uma noção básica de como deve ser implementado a interface. Com a criação dos Casos de Uso, a fase de Concepção é finalizada, seguindo para a fase de Elaboração e Construção.

4.3 Expansão dos Casos de Uso

Com o fim da fase de concepção do projeto, os requisitos foram levantados assim como os casos de uso do sistema definidos. Portanto, pode-se começar a fase de elaboração e construção da UP, onde se executa uma investigação mais detalhada da fase anterior.

Essa etapa é iniciado com a Expansão dos Casos de Uso, que tem como finalidade o aprofundamento da análise dos requisitos levantando anteriormente, especificando os casos de uso detalhadamente, seus fluxos principais e alternativos, descrevendo como o sistema se comporta naturalmente e quais caminhos ele pode tomar dependendo das ações do usuário. Por fim, mostra-se as exceções, que definem ações do usuário que criam caminhos não convencionais no fluxo principal ou alternativo.

Na Tabela 3 é apresentada a expansão do caso de uso *Configurar Controle*,

Tabela 3 – Expansão do caso de uso Configurar Controle

Fluxo Principal	**Fluxos Alternativos**
1. O usuário cria um controlador do tipo MPC, escolhendo o tipo do controlador preditivo (GPC, PNMPC ou DMC). 2. O usuário define os parâmetros do controlador criado anteriormente. 3. As restrições do controlador são definidas pelo usuário, caso necessário. 4. O modelo de predição que o MPC irá se basear para os cálculos da lei de controle é definido pelo usuário.	1a. Se o modelo é do tipo função transferência, o algoritmo de controle é o GPC. 1b. Se o modelo é do tipo equação de diferença, então o algoritmo de controle é o PNMPC. 1c. Se o modelo é do tipo resposta ao degrau, então o algoritmo de controle é o DMC. 2a. Usuário pode copiar um controlador existente, criando uma réplica do controlador escolhido. **Exceções** 1a. Dois controladores não podem ter o mesmo nome, gerar janela de erro se o usuário criar. 1b. O usuário não pode dar nomes vazios para os controladores. 2a. Controlador copiado não pode ter mesmo nome que outro controlador já existente.

descrevendo com mais detalhes o fluxo principal, os alternativos e também as exceções. O fluxo principal indica ações que são tomadas durante o funcionamento padrão do sistema, modelando o seu comportamento mais comum. O fluxo alternativo indica situações menos convencionais, mas que não necessariamente indicam mau funcionamento do sistema. Por fim, as exceções indicam ações que resultam no travamento do sistema, podendo causar problemas ao funcionamento dele caso não sejam tratadas devidamente.

4.4 Modelo conceitual

Nessa etapa é criada uma representação lógica do sistema, de alto nível, abstendo-se de qualquer análise do software que será utilizada para implementação.

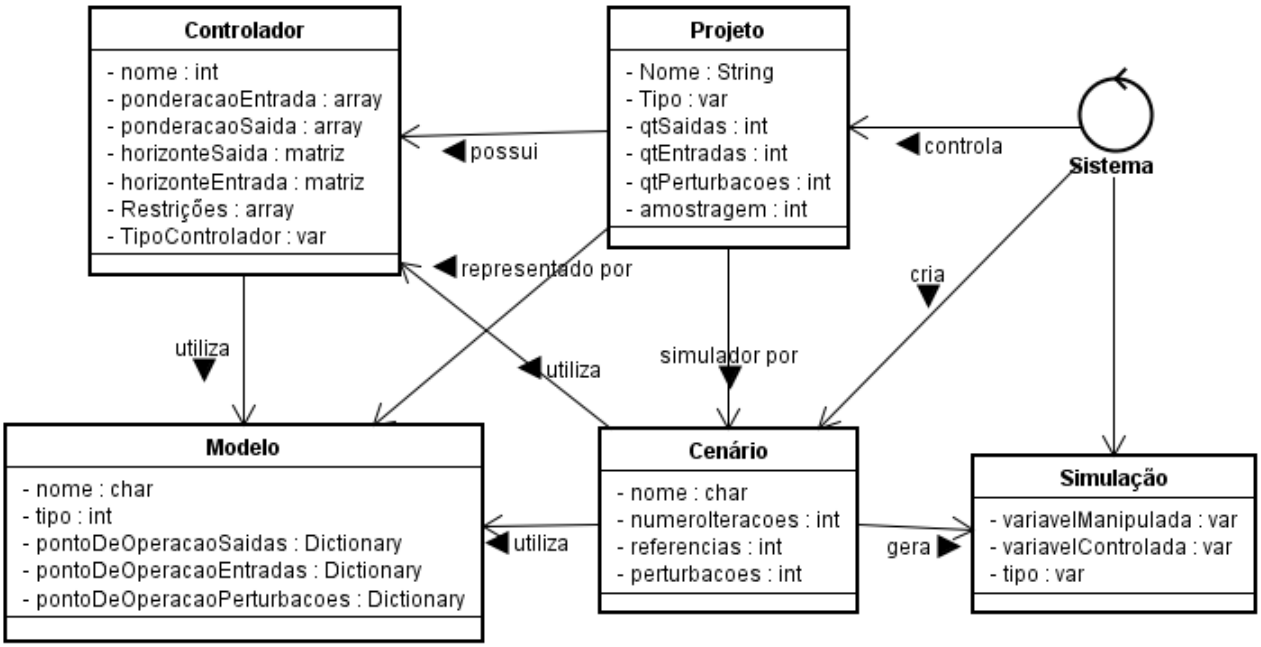

Figura 15 – Modelo conceitual do projeto.

Portanto, são criados conceitos para refletir os casos de uso expandidos anteriormente sem fazer referências às operações ou outros aspectos dinâmicos do sistema projetado [27]

O Diagrama Conceitual proposto pode ser observado na Figura 15, definindo os conceitos como:

- Sistema: Representação do software na sua totalidade, não um conceito propriamente dito;
- Projeto: Conceito utilizado que irá armazenar as principais informações do projeto, como por exemplo uma lista contendo todos os modelos do sistema. Seus atributos são nome, tipo (Discreto), quantidade de saídas, entradas e perturbações, lista dos modelos, controladores e cenários, lista das variáveis do processo e pôr fim a amostragem;
- Modelo: O modelo é uma representação matemática do processo. Seus atributos são nome, tipo (Função Transferência/Equações de Diferença/Resposta ao Degrau) e os pontos de operações;
- Controlador: Representa o algoritmo que calcula a ação de controle utilizando o MPC, de forma a controlar o processo. Seus atributos são os horizontes, as ponderações, as restrições e o tipo do controlador (GPC/PNMPC/DMC);
- Cenário: Conceito que realiza toda a configuração do cenário de simulação, utilizando os modelos e controladores para criar uma simulação, selecionando as

referências e perturbações. Seus atributos são nome, referências, perturbações e o número de iterações, ou seja, o tempo de simulação;

- Simulação: Representa o resultado obtido ao criar e executar o cenário, gerando gráficos com as variáveis manipuladas e controladas. Seus atributos são as resposta das variáveis manipuladas e controladas, e o tipo (Malha Aberta/Malha Fechada).

4.5 Contratos e Diagrama de Sequência

Partindo dos casos de uso encontrados anteriormente, pode-se definir as operações e consultas do sistema. Os contratos permitem representá-las identificando as pré-condições, as pós-condições e por fim as exceções para cada operação e consulta realizada. A principal importância dos contratos está na descrição da responsabilidade de cada classe, atentando às mudanças do estado dos objetos quando alguma operação ou consulta do sistema são executadas, descrevendo *o quê* deve ser feito, sem se preocupar em *como* deve ser feito.

Tabela 4 – Contrato de uma das funções da Interface Gráfica

Operação: Escolhe_Cenario(cenario) **Pré-Condição**: Existir um objeto cenário válido com nome não-vazio. **Pós-Condição**: O objeto cenário foi retornado, de forma que a interface poderá realizar a simulação usando as informações desse cenário, podendo também alterar as suas informações.

Na Tabela 4 aparece a descrição da operação Escolhe_Cenario, que basicamente seleciona um cenário de simulação, mostrando que para ocorrer essa operação existe uma pré-condição e uma pós-condição para sua execução. Nesse caso, deve haver algum cenário já criado pelo usuário com nome não-vazio para posteriormente retornar o objeto que representa o cenário escolhido para que o usuário possa alterá-lo também permitindo a sua simulação.

É muito comum utilizar os diagramas de sequência junto aos contratos para representar uma sequência de processos realizadas pelo usuário, cumprindo com os contratos definidos para cada operação para executá-las. Seu principal objetivo é mostrar interações entre diferentes objetos a partir de casos de uso, representando uma linha de tempo no qual eventos podem ocorrer.

O diagramas de sequência mostrado na Figura 16 mostra como ocorre a simulação de um cenário na interface gráfica. O que pode ser observado no diagrama é:

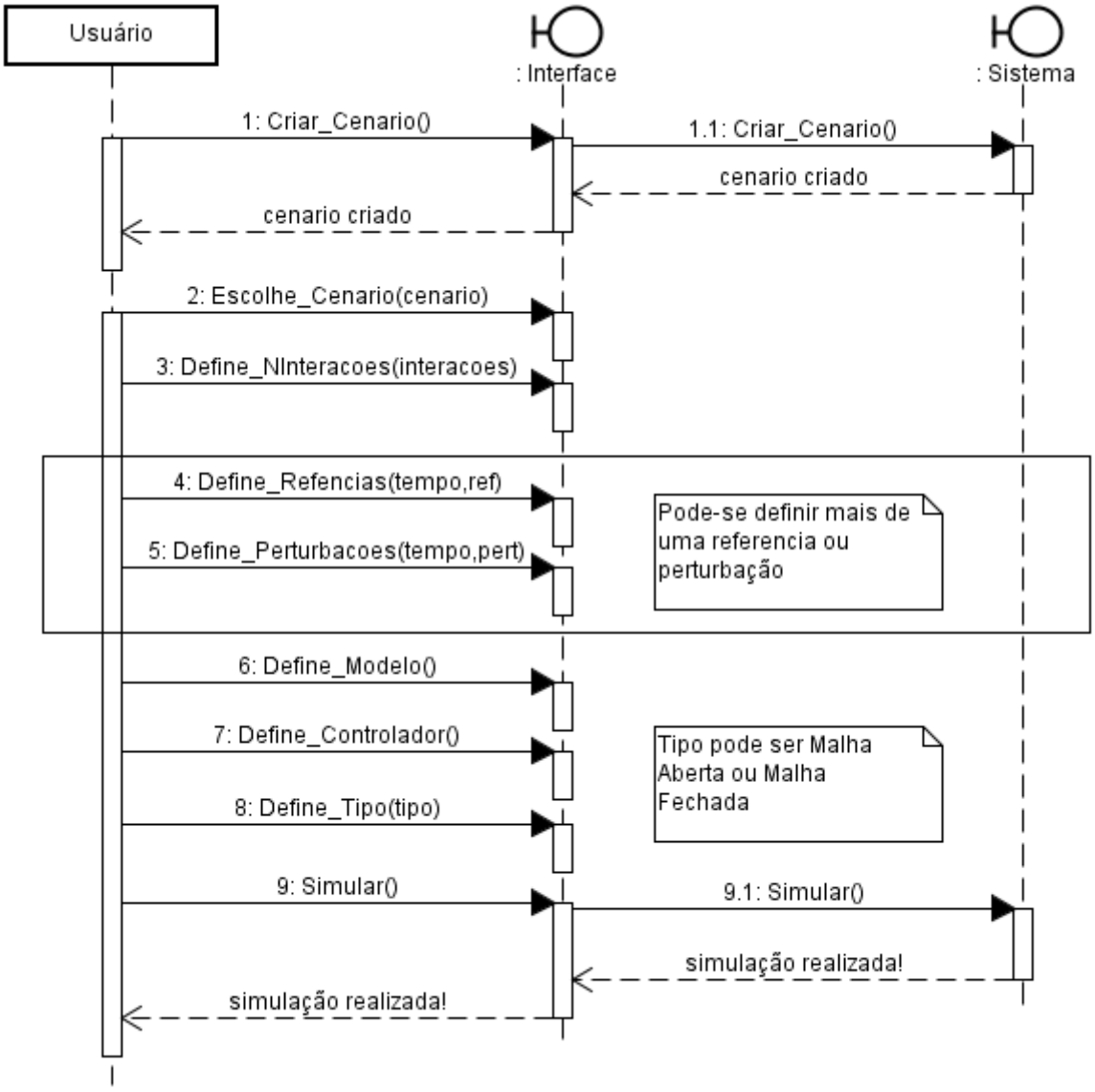

Figura 16 – Diagrama de sequência representando a simulação de um cenário.

1. O objeto cenário é criado executando a operação Criar_Cenario() e o sistema retorna um aviso que o cenário foi criado;
2. O usuário seleciona um cenário para ser simulado, chamando a função Escolhe_Cenario(cenario), descrita no contrato da Tabela 4 e que recebe o cenário criado anteriormente como parâmetro de execução;
3. O usuário define o número de iterações necessárias durante a simulação;
4. São escolhidas as referências e perturbações aplicadas ao processo, sendo permitido definir mais de uma para referência ou perturbação;
5. É definido o modelo e o controlador da simulação;
6. Usuário define o tipo da simulação, se será de malha aberta ou malha fechada;

7. A simulação de fato é realizada, chamando a função Simular(), retornando um aviso de que a operação foi realizada com sucesso.

4.6 Diagrama de Colaboração e de Classes

Nessa etapa do projeto de software, busca-se uma solução ao problema identificado durante a etapa de elaboração do projeto. A parte do projeto lógico visa ampliar o que foi desenvolvido no modelo conceitual, nos contratos e diagramas de sequência de forma resultando nos diagramas de colaboração e de classes, sempre pensando já na realização do software propriamente dito. O projeto lógico tenta resolver os problemas na camada de domínio da aplicação, independente da tecnologia que desenvolverá o sistema.

Os diagramas de colaboração são criados a partir dos diagramas de sequência e determinam com detalhes a troca de mensagens entre instâncias de objetos das diversas classes projetadas.

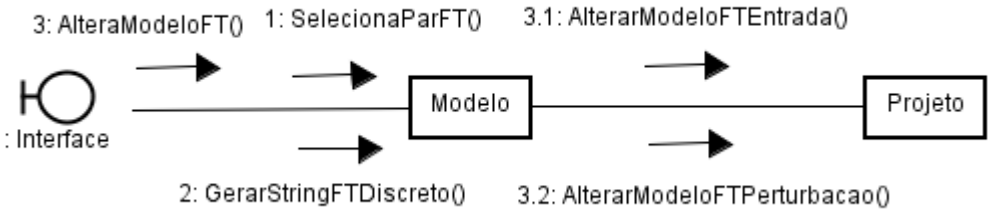

Figura 17 – Diagrama de colaboração para a função AlterarFT.

Na Figura 17 é observado um diagrama de colaboração para uma função que tem como objetivo alterar uma Função Transferência de uma classe Modelo. Inicialmente são escolhidos os pares entrada/saída ou perturbação/saída que irão compor a função transferência a ser modificada, utilizando a função SelecionarParFT(). Depois, a partir da interface, é determinada a nova função transferência com a função GerarStringFTDiscreto(). Por fim, a interface altera a função e envia uma mensagem, AlterarModeloFT(), à classe Projeto para ela identificar se o par alterado era entrada/saída ou perturbação/saída, alterar a função transferência para esse par de variáveis e armazenar a informação.

O diagrama de classes do projeto atualmente é representado na Figura 18, sem os atributos e métodos correspondentes para facilitar a visualização do diagrama. Como citado no capítulo anterior, o diagrama primeiramente é uma cópia do modelo conceitual, para então acrescentar mais informações ao modelo, adicionando métodos, atributos e aumentando a complexidade da representação usando o conceito de herança. Ela

permite representar o sistema de forma simples, sem uso excessivo de informação. Por exemplo, a classe Modelo apresenta três subclasses que herdam suas características pois essas três subclasses (ModeloFT, ModeloNLinear e ModeloRespostaDegrau), possuem atributos e métodos similares, como nome e tipo do modelo, número de entradas e saídas, e etc. Porém, alguns atributos são únicos à classe, como no caso do ModeloFT que utiliza funções transferências para representá-los, o que exige a diferenciação entre subclasses para uma implementação coerente.

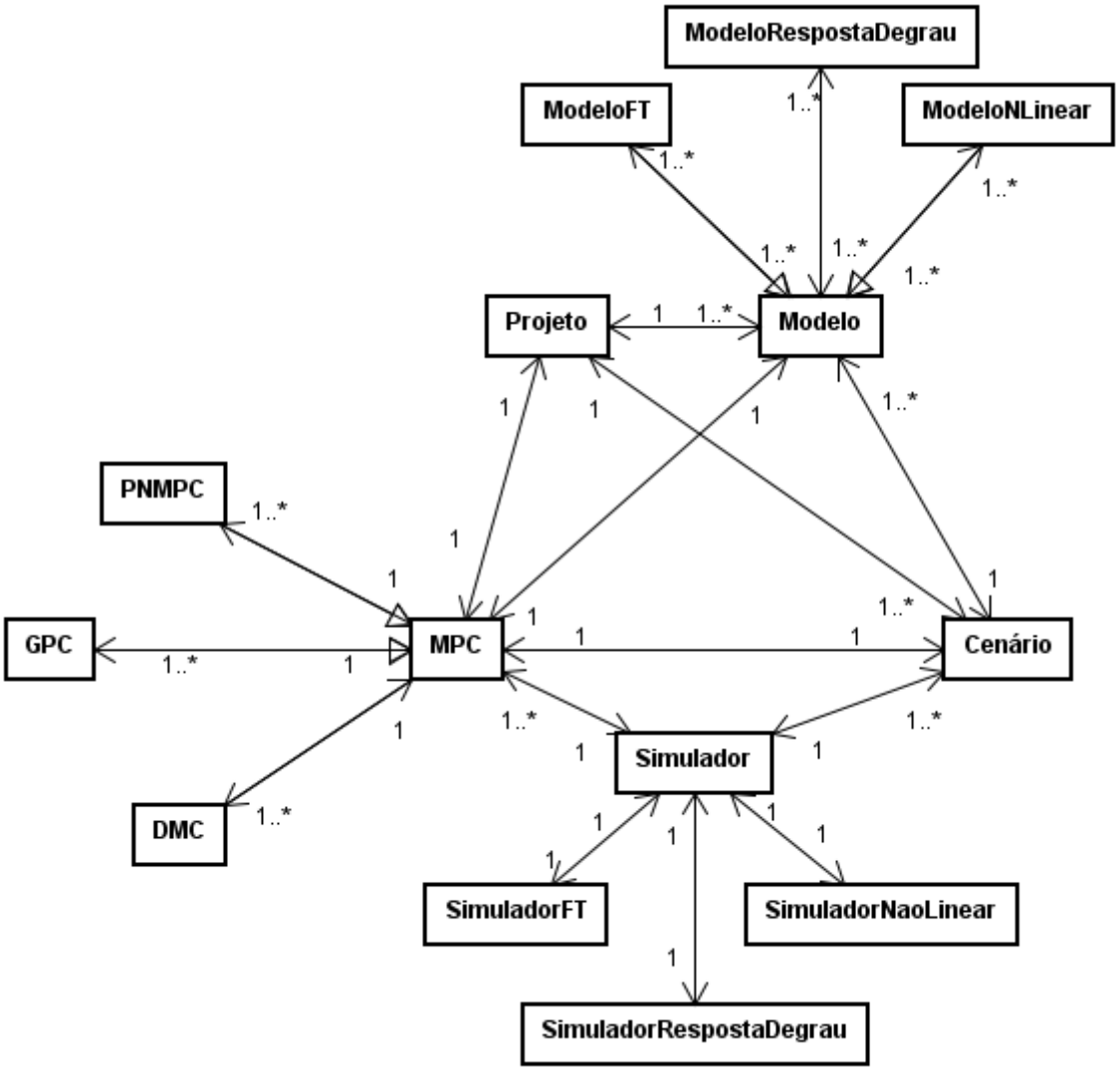

Figura 18 – Diagrama de classes do sistema.

As principais classes do projeto são mostradas abaixo com uma breve descrição dos seus atributos e métodos mais importantes.

Classe Projeto

A classe Projeto contém atributos e métodos necessários para deixar a interface operacional, armazenando todas as informações básicas referentes aos modelos,

controladores, cenários e as variáveis do projeto. Além disso, tem um papel de gerenciamento, permitindo alterações na simulação, no cenários, também com métodos importantes como SimulaCenario(), que realiza toda a simulação, seja ela de malha aberta ou fechada.

Classe Modelo

A classe Modelo e suas subclasses tem como principal objetivo criar os modelos do processo a ser simulado, utilizando duas estruturas principais: Funções Transferências e Modelo de Resposta ao Degrau para a simulação de sistemas lineares e Equações de Diferenças para sistemas não-lineares. Portanto, nela são armazenadas as informações relacionadas aos modelos.

Classe Controlador

Semelhante à ideia utilizada na classe Modelo, a classe Controlador e as subclasses armazenam dados referentes aos controladores usados durante a simulação. Os tipos de controladores são: *Generalized Predictive Controller* (GPC) e *Dynamic Matrix Control* (DMC) para controle preditivo linear e *Practical Non-Linear Predictive Controller* (PNMPC) para controle preditivo não-linear. Então, os parâmetros dos controladores, as restrições de controle, os modelos de predição e outros dados são guardados para uso pela interface, além de atributos que calculam a ação de controle e realizam os algoritmos necessários para o controle, seja linear ou não-linear. A classe controlador e a classe modelo apresentam uma associação, visto que os controladores preditivos necessitam de modelos para encontrar as predições e, posteriormente, encontrar as ações de controle.

Classe Cenário

Na classe Cenário, o sistema guarda informações referentes a simulação, como as referências, perturbações, o tempo de simulação e também o tipo de simulação, que pode ser de malha aberta ou fechada. Alguns atributos, como setControlador(), tem como função definir as características gerais da simulação, como qual o modelo a ser simulado, qual o controlador, os valores atuais das saídas e entradas, etc.

Classe Simulador

Por fim, a classe Simulador tem como função prover os dados de necessários para a interface gerar os resultados da simulação, ou seja, para que os gráficos da simulação sejam fornecidos ao usuário. Os principais atributos são os cenários, as entradas, saídas calculadas, perturbações, as referências e os métodos mais importantes

são SalvarDados() que armazena os valores de todas as variáveis em cada instante de tempo e também SimulaCenario() que faz os cálculos propriamente dito das saídas, invocando funções da classe Controlador para o cálculo das ações de controle.

4.7 Implementação

Com a finalização do desenvolvimento da camada de domínio, o projeto lógico é criado e pode-se começar a geração de códigos. A partir disso, são gerados os códigos das classes correspondentes a camada de domínio da aplicação com as respectivas operações e consultas de cada classe definida anteriormente [3].

Portanto, durante a etapa atual o projeto já está sendo implementado de fato, a geração de códigos de programação foca principalmente nas classes definidas na etapa anterior (ver Figura 18). O diagrama de estados visa representar o funcionamento dos objetos das classes do sistema desenvolvido no decorrer da execução do software.

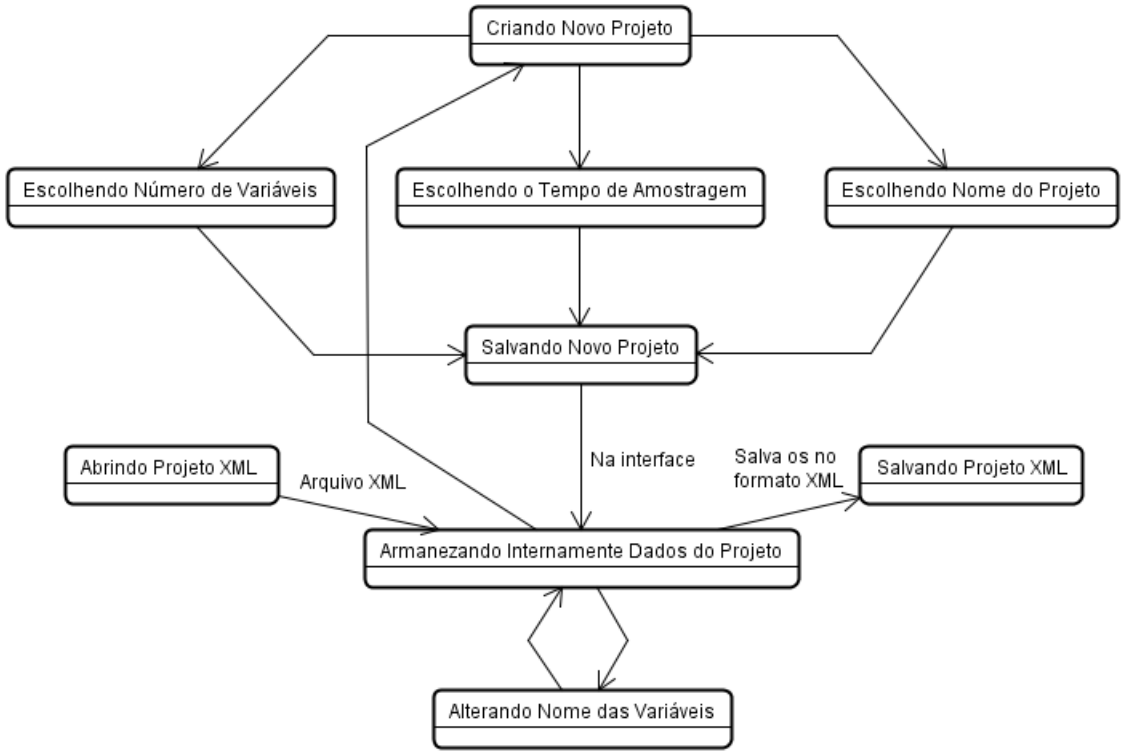

Figura 19 – Diagrama de estados da classe Projeto.

Um exemplo está demonstrado na Figura 19, que consiste no diagrama de estados referente à classe Projeto de forma simplificada, descrevendo os comportamentos e as possíveis situações de um objeto da classe.

Observa-se que, inicialmente, o usuário cria um novo projeto iniciando o sistema e escolhendo os parâmetros básicos para que os dados sejam armazenados interna-

mente, sendo possível alterar o nome *default* definido para as variáveis de entrada e saída após a inicialização do projeto. O projeto também pode ser inicializado abrindo um arquivo XML armazenado na camada de persistência, caso existam dados armazenados. Além disso, o usuário pode guardar os dados criados, salvando-os na camada de persistência, de maneira que o usuário consiga recuperar os dados posteriormente. Além disso, caso necessário, pode-se recomeçar um projeto novo que reinicializa todo o processo descrito, mantendo a mesma lógica.

4.7.1 Qt Designer

A interface foi criada utilizando a ferramenta Qt Designer e posteriormente traduzida para a linguagem Python utilizando comandos pré-determinados, devido a grande compatibilidade entre o Python e o Qt. A Figura 20 mostra como o Qt Designer foi utilizado para criar a interface, sendo uma ferramenta simples e de fácil utilização.

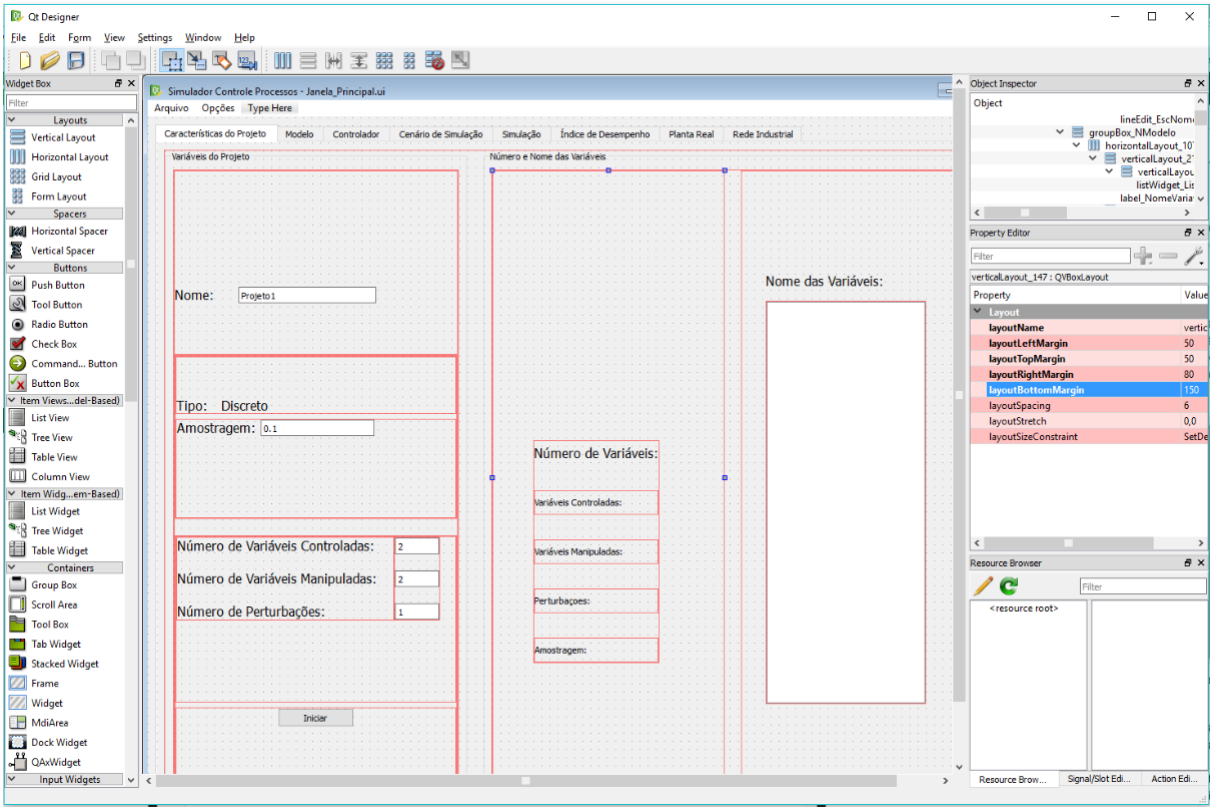

Figura 20 – Qt Designer.

Na parte esquerda do Qt Designer estão as diferentes opções de estrutura que podem ser utilizadas para o desenvolvimento da interface, que são elementos básicos mas de grande importância, como botões, textos, linhas e etc. No canto direito aparecem as definições dos nomes dos elementos da interface, além de outros parâmetros que contribuem para coerência e design do projeto. O Qt Designer utiliza o conceito de classes para definir cada estrutura, seja um simples botão até uma aba

completa com todos elementos dispostos, onde cada elemento se torna uma instância de objeto, se valendo do mesmo conceito de herança definido anteriormente.

A extensão de arquivo adotada pelo Qt Designer é .ui. Desta forma, para que seja possível a utilização da interface utilizando o Python é necessária a conversão do arquivo .ui para .py, ou seja, a extensão utilizada pelo Qt Designer deve ser adaptada para que possa ser interpretada na linguagem Python, que apenas lida com arquivos .py. Para tal, utiliza-se o comando abaixo para a conversão das extensões:

pyuic4 interface.ui -o ..\interface.py

Com o arquivo convertido, pode-se começar a criar algoritmos que façam a simulação de fato na própria API do Python, facilitando a implementação do código e criação do software.

4.8 Camada de Persistência

A primeira análise que foi feita antes de se iniciar o desenvolvimento da camada de persistência foi quais seriam os dados que deveriam ser armazenados e de que tipo eles seriam. A camada de persistência deve ser capaz de guardar todos os dados necessários de forma que eles, ao serem recuperados, permitam que o projeto seja recriado exatamente igual e por isso é muito importante decidir quais as informações mais relevantes do sistema.

```
<!--Projeto-->
<xs:element name="Projeto">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="nomeProjeto" type="xs:string"/>
            <xs:element name="tipoProjeto" type="enumProjeto"/>
            <xs:element name="qtEntradas" type="xs:nonNegativeInteger"/>
            <xs:element name="qtSaidas" type="xs:nonNegativeInteger"/>
            <xs:element name="qtPerts" type="xs:nonNegativeInteger"/>
            <xs:element name="Amostragem" type="xs:float"/>
            <xs:element name="listaEntradas" type="xs:string"/>
            <xs:element name="listaSaidas" type="xs:string"/>
            <xs:element name="listaPerturbacao" type="xs:string"/>
            <xs:element name="listaModelos" type="listaModelos"/>
            <xs:element name="listaControladores" type="listaControladores"/>
            <xs:element name="listaCenarios" type="listaCenarios"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
```

Figura 21 – Fragmento do *schema* criado para o projeto.

Inicialmente foi criado o arquivo *schema* que continha a estrutura básica do documento XML que seria gerado na camada de persistência, informando detalha-

damente os tipos e características das informações. Na Figura 21 é mostrado um fragmento do arquivo que define as características do *schema*, criando um elemento complexo composto por uma sequência de outros elementos, cada um de um tipo específico. Esses elementos compõem as informações que serão armazenadas na camada de persistência e, portanto, são suficientes para que o projeto possa ser recuperado posteriormente.

Com o *schema* definindo a estrutura dos documentos XML, foi então implementado o código que permitiria ao usuário na própria interface gráfica salvar dados de um projeto criado. Para interpretar os arquivos XML é comum utilizar pacotes ou bibliotecas integradas à linguagem de programação, e no caso do Python existem diversas bibliotecas que permitem a leitura e escrita de arquivos XML. Entre as opções disponíveis foi utilizada a biblioteca *PyXB* [36] pela simplicidade em integrar código do Python com o arquivo *schema*. Na Figura 22 é observado um exemplo de XML gerado, seguindo o padrão do *schema* mostrado, representando o projeto de um tanque cônico hipotético.

```
<?xml version="1.0" ?>
<Projeto>
    <nomeProjeto>Tanque Conico</nomeProjeto>
    <tipoProjeto>Discreto</tipoProjeto>
    <qtEntradas>1</qtEntradas>
    <qtSaidas>1</qtSaidas>
    <qtPerts>1</qtPerts>
    <Amostragem>0.1</Amostragem>
    <listaEntradas>{0: 'qe'}</listaEntradas>
    <listaSaidas>{0: 'L'}</listaSaidas>
    <listaPerturbacao>{0: 'qs'}</listaPerturbacao>
    <listaModelos>
        <modeloNaoLinear>
            <nomeModelo>Tanque Conico</nomeModelo>
            <tipoModelo>NL</tipoModelo>
            <pontoDeOperacaoSaidas>{0: 0.0}</pontoDeOperacaoSaidas>
            <pontoDeOperacaoEntradas>{0: 0.0}</pontoDeOperacaoEntradas>
            <pontoDeOperacaoPerturbacoes>{0: 0.0}</pontoDeOperacaoPerturbacoes>
            <saturacaoMax>{0: 9.85}</saturacaoMax>
            <saturacaoMin>{0: 0.0}</saturacaoMin>
            <estadoSaturacao>{0: 1}</estadoSaturacao>
            <equacaoCerta>['L[k-1] + 0.1*((((H**2)/(pi*R**2))*
                                (qe[k-1] - qs[k-1]))**(1/3))']</equacaoCerta>
            <listaConstantes>{0: 'H',1: 'R'}</listaConstantes>
            <constantes>{'H': 9.85,'R': 4.5}</constantes>
        </modeloNaoLinear>
    </listaModelos>
    <listaControladores/>
    <listaCenarios/>
</Projeto>
```

Figura 22 – Documento XML gerado para o projeto de um tanque cônico.

Os arquivos gerados são suficiente para que o usuário consiga *Salvar*, *Salvar*

Como e *Abrir* os projetos desenvolvidos, permitindo o desenvolvimento de um banco de dados com os diversos projeto criados. Para facilitar a criação desse banco de dados, foi implementada uma biblioteca que contém projetos com modelos comuns da indústria, com intuito de criar uma base de dados sólida para o usuário antes mesmo dele iniciar o uso da interface gráfica. Porém, alguns cuidados foram tomados para que as informações armazenadas não fossem corrompidas durante alguma simulação, protegendo todos os dados guardados nessa biblioteca, de forma que ela estivesse fora do alcance do usuário. Por isso, os arquivos da biblioteca foram criptografados e salvos em arquivos de texto inacessíveis ao usuário.

Considerações Finais

Nesse capítulo foram apresentadas as etapas da metodologia para desenvolver o software proposto, detalhando os aspectos mais importantes do desenvolvimento. Muitas etapas foram realizadas durante o início do projeto, porém algumas foram atualizadas para se adequar à versão final. Por exemplo, quando o projeto começou não estava prevista a aplicação do algoritmo DMC para controle preditivo e modelos do tipo resposta ao degrau, tendo que adaptar os diagramas de classe do projeto. Também não são mencionados com detalhe os controladores PID que não são foco do sistema, pois foram implementados apenas como uma ferramenta adicional de simulação.

5 Interface de Usuário

Neste capítulo é apresentado o resultado final da interface desenvolvida, detalhando cada característica e explicando no formato de um pequeno manual demonstrando o funcionamento da interface gráfica. Alguns elementos implementados na interface não são explicados aqui pois não foram implementados para esse projeto, mas para outros que aconteciam em paralelo.

O software criado, portanto, implementa uma interface gráfica, chamada atualmente de *SimPro*, com intuito de simular processos industriais e com ênfase em controle preditivo. Todo o processo de simulação foi criado de forma que seguisse uma sequência lógica básica de operações, buscando compreender como são feitos testes e simulações de processos industriais na prática para aprimorar o uso da interface. Com isso foram analisados as classes do sistema para definir qual seria essa sequência lógica básica, obtendo:

1. Primeiramente o usuário deve iniciar o *projeto* definindo suas *características*;
2. Depois, todos os *modelos* que serão utilizados na simulação devem ser definidos;
3. Então, o usuário define a sintonia dos *controladores*;
4. Na sequência são criados os *cenários* de simulação e a *simulação* propriamente dita é realizada;
5. Com a simulação realizada, o usuário verifica os resultados da *simulação* a partir de gráficos;
6. Por fim, o usuário pode explorar a robustez dos controladores definidos analisando os *índices de desempenho*.

Considerando essa sequência, pode-se observar que existem seis ações principais que o usuário pode executar durante a simulação, que são distintas e apresentam características únicas. Com o intuito de aproveitar esse fato constatado, a interface gráfica foi criada com seis abas distintas, cada uma representando um dos elementos principais destacados na sequência lógica exposta: as *Características do Projeto*, o *Modelo*, o *Controlador*, o *Cenário*, a *Simulação* e os *Índices de Desempenho*. Na Figura 23 está a versão final da interface implementada com as seis abas (mais duas abas que, como dito anteriormente, não são escopo desse projeto) e uma barra de ferramentas.

A implementação da barra de ferramentas teve como objetivo possibilitar ao usuário o acesso rápido a certas funções especiais do sistema, que acrescentam

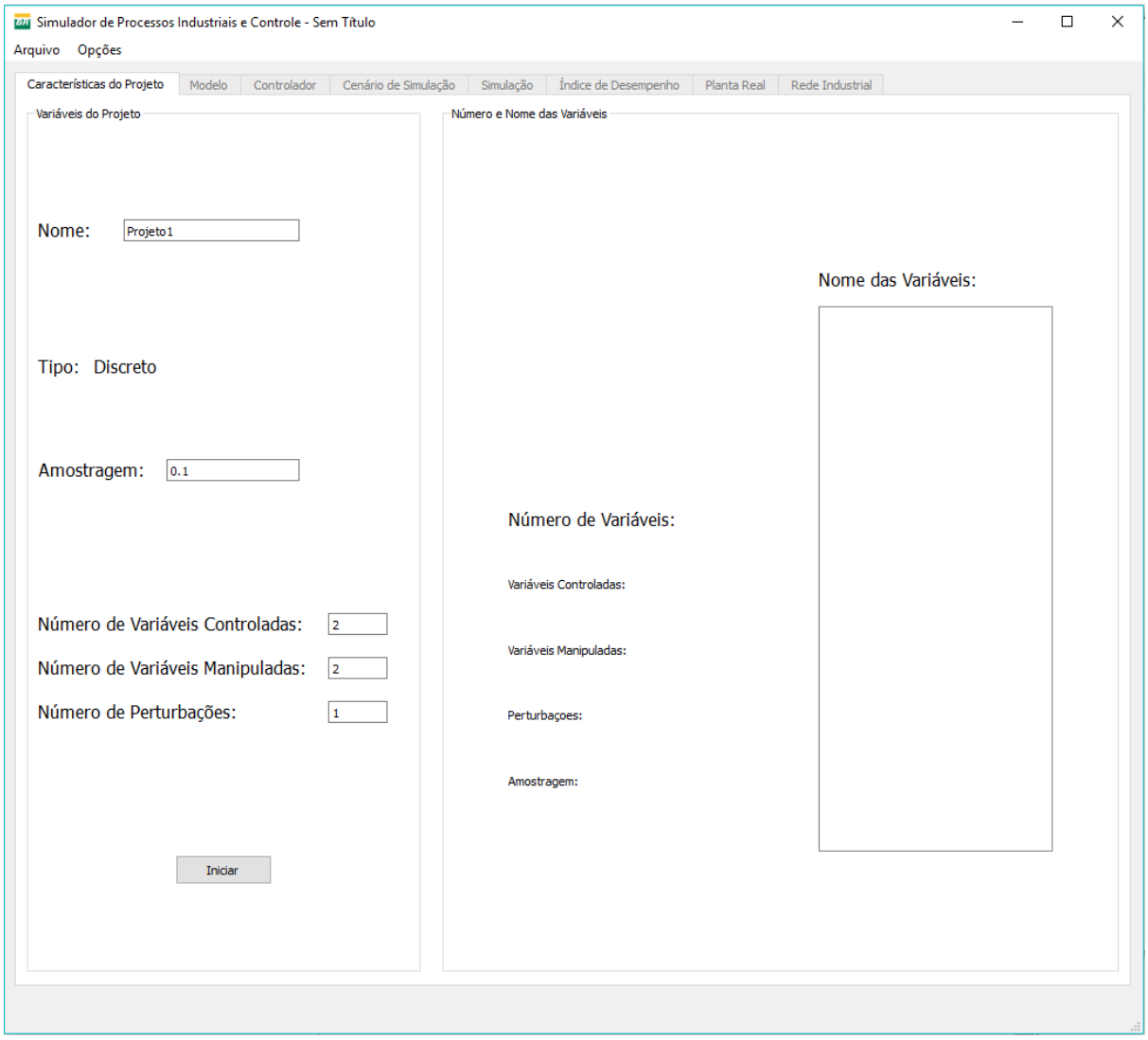

Figura 23 – Janela principal da interface com as seis abas definidas.

características a ele que melhoram sua performance e sua usabilidade. Na parte do *Arquivo* na barra de ferramenta, essas funções são (ver Figura 24):

- *Novo...*: Permite ao usuário reiniciar um projeto em andamento, retornando o sistema ao seu estado inicial;
- *Abrir...*: Essa função possibilita abrir arquivos XML que estão armazenados no computador e que tenham a mesma estrutura definida pelo *schema*;
- *Salvar*: Caso o usuário tenha alterado o arquivo a função Salvar permite que os dados alterados sejam armazenados num arquivo XML, de forma que ele possa recuperar o projeto criado;
- *Salvar Como...*: Semelhante à Salvar, porém o usuário também define o nome do arquivo XML gerado:

- *Sair*: Finaliza o projeto, perguntando ao usuário se ele deseja salvar caso alguma alteração seja realizada no projeto.

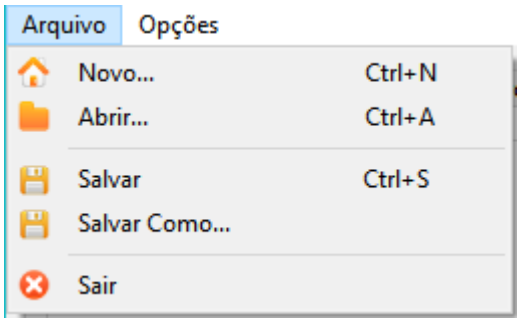

Figura 24 – Arquivo na barra de ferramentas.

Para facilitar o uso da barra de ferramentas, algumas das funções foram implementadas com *atalhos*, diminuindo esforço e o tempo gasto utilizando a interface. O outro elemento da barra de ferramentas é *Opções*, que basicamente implementa a biblioteca de modelos (Figura 25) estudada no final do capítulo.

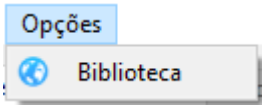

Figura 25 – Opções na barra de ferramentas.

Nas seções seguintes serão examinadas cada aba da interface gráfica com detalhes, explicando o funcionamento de cada atributo da aba.

5.1 Características do Projeto

Cada aba tem uma funcionalidade particular dentro da sequência lógica citada anteriormente. Essa sequência também pode ser considerada cronológica, na maioria das simulações realizadas, pois algumas ações tomadas pelo usuário seguem uma ordem específica. Por exemplo, o usuário só pode criar um modelo do processo se o projeto for iniciado, definindo o número de variáveis manipuladas, controladas e perturbações do sistema. Para resolver esse problema, todas as abas se iniciam desabilitadas, exceto a aba *Características do Projeto* (ver Figura 23) e elas só serão habilitadas quando o usuário iniciar o projeto, determinando as características dele.

Num primeiro instante, portanto, o usuário define os parâmetros do projeto. Na Figura 26 estão definidas as variáveis do projeto:

- *Nome*: define o nome dado ao projeto. Não existe nenhuma restrição quanto ao nome, é sugerido nomes que tenham relação com o escopo da simulação, como

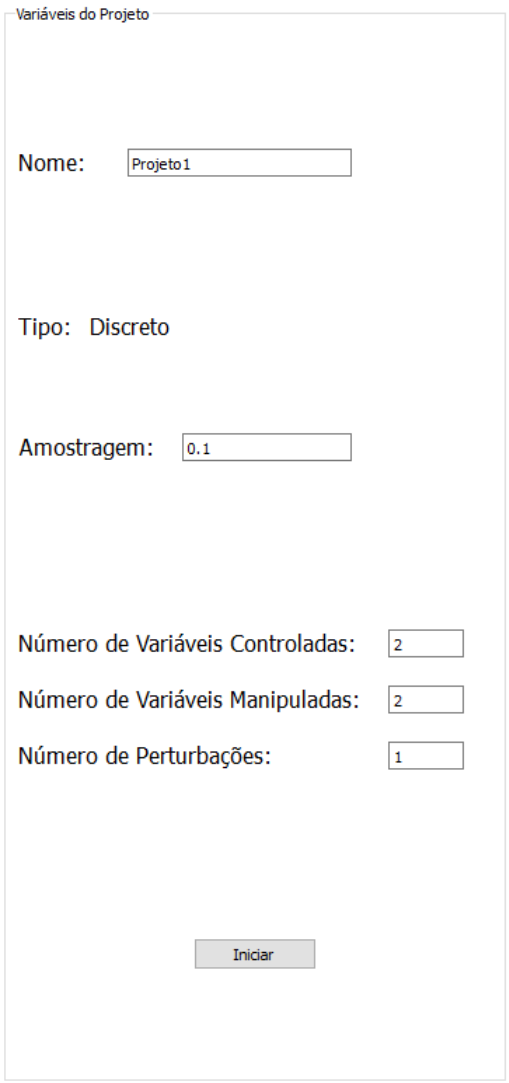

Figura 26 – Variáveis do projeto.

o processo que será simulado, ou também nomes que identificam especificadamente qual o projeto está sendo feito;

- *Tipo*: A princípio, a simulação poderia ser feita em sistemas discretos e contínuos, porém foi decidido que seriam implementados apenas sistemas discretos, visto que atualmente a maioria dos processos são controlados digitalmente. Portanto, o processo é sempre representado em sistemas do tipo *Discreto*;

- *Amostragem*: os sistemas discretos são amostrados e, portanto, também deve ser definido um tempo de amostragem do processo, permitindo apenas valores racionais positivos.

- *Número de Variáveis Controladas*: define o número de saídas do sistema, permitindo apenas número inteiros positivos.

- *Número de Variáveis Manipuladas*: define o número de entradas do sistema, permitindo apenas número inteiros positivos.
- *Número de Perturbações*: define o número de perturbações do sistema, permitindo apenas número inteiros positivos.

O software permite a implementação de sistemas monovariáveis ou multivariáveis, definidos de acordo com a necessidade do usuário que escolhe se o sistema será SISO ou MIMO. Caso o usuário tente definir algum dos parâmetros de forma inválida, uma janela de aviso aparecerá para informá-lo de que a operação foi inválida. Por exemplo, se for escolhido um número de entradas igual a 2.5, um aviso será gerado como o da Figura 27, notificando o usuário qual operação foi realizada de maneira incorreta e impedindo que o projeto seja iniciado até que todos parâmetros sejam definidos corretamente.

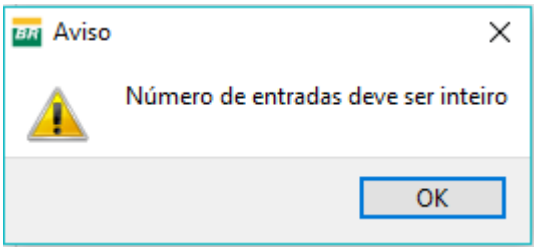

Figura 27 – Aviso gerado quando o número de entradas escolhidas for inválido.

Também foi aproveitado um elemento disponível no Qt Designer para auxiliar o usuário quando ele está usando a interface gráfica, chamado de *ToolTip*. O seu intuito é dar dicas ao usuário sobre como utilizar a interface e os seus atributos, sem atrapalhar a visualização. Para aparecer o *ToolTip* basta deixar a seta do *mouse* estática em cima de algum elemento da interface que automaticamente irá aparecer um texto revelando alguma informação útil.

Após o usuário definir as variáveis do projeto corretamente, o projeto pode ser iniciado clicando no botão *Iniciar*. Em seguida, os parâmetros escolhidos são mostrados na própria aba e uma lista é criada com o nome das variáveis do processo, conforme a Figura 28.

O *padrão* do sistema é definir as saídas como 'Y', as entradas como 'U' e as perturbações como 'Q', porém o próprio usuário pode alterar o nome das variáveis, clicando duas vezes na variável que deseja alterar da lista, como mostrado na Figura 29. O nome das variáveis sempre dependerá do processo em questão, por exemplo, num sistema de controle de nível que utiliza-se válvulas para manipular a vazão de entrada seria lógico definir a variável controlada como 'Nível' e a manipulada como 'Vazão de Entrada'. A escolha dos nomes padrões surgiu dos nomes mais comuns

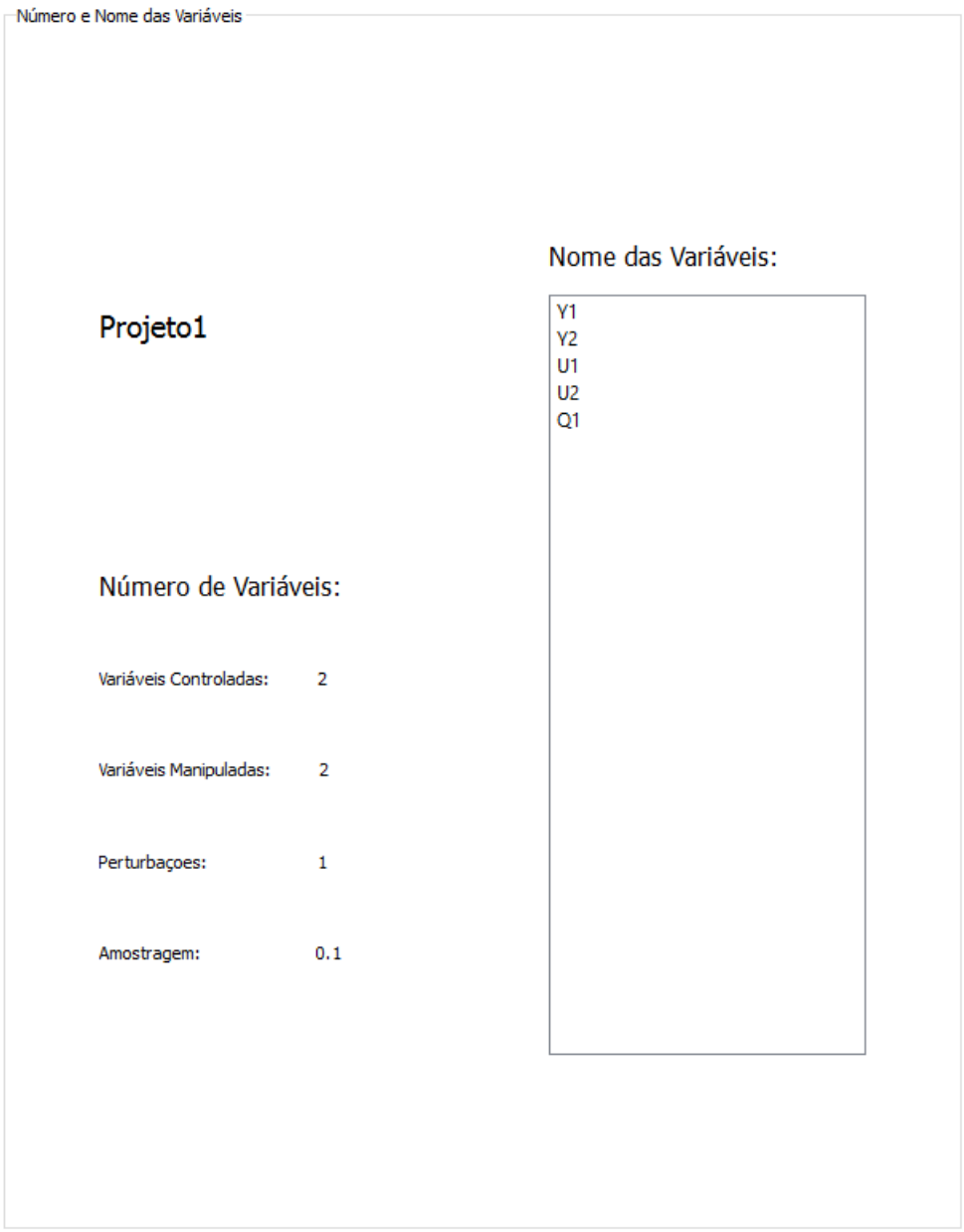

Figura 28 – Dados do projeto e nome das variáveis.

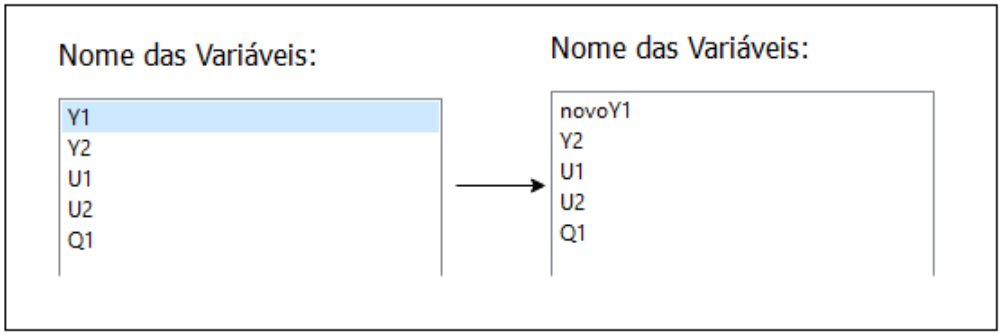

Figura 29 – Demonstração de como modificar nome da variável.

encontrados nas literaturas relacionadas a controle de processos para representar sistemas genéricos.

Com as características básicas do projeto definidas, é permitido ao usuário navegar entre as outras abas da interface. Seguindo a lógica criada, o próximo passo é a definição dos modelos do processo na aba seguinte.

5.2 Modelo

Na aba *Modelo* são configurados todos os modelos do processo que o usuário irá utilizar durante a simulação, incluindo os modelos utilizados pelos controladores. Na Figura 30 pode-se observar as diversas funcionalidades da aba e como estão organizados os diferentes tipos de modelo. Ela foi criada para se adaptar ao tipo de modelo escolhido, modificando sua aparência dependendo do tipo do modelo.

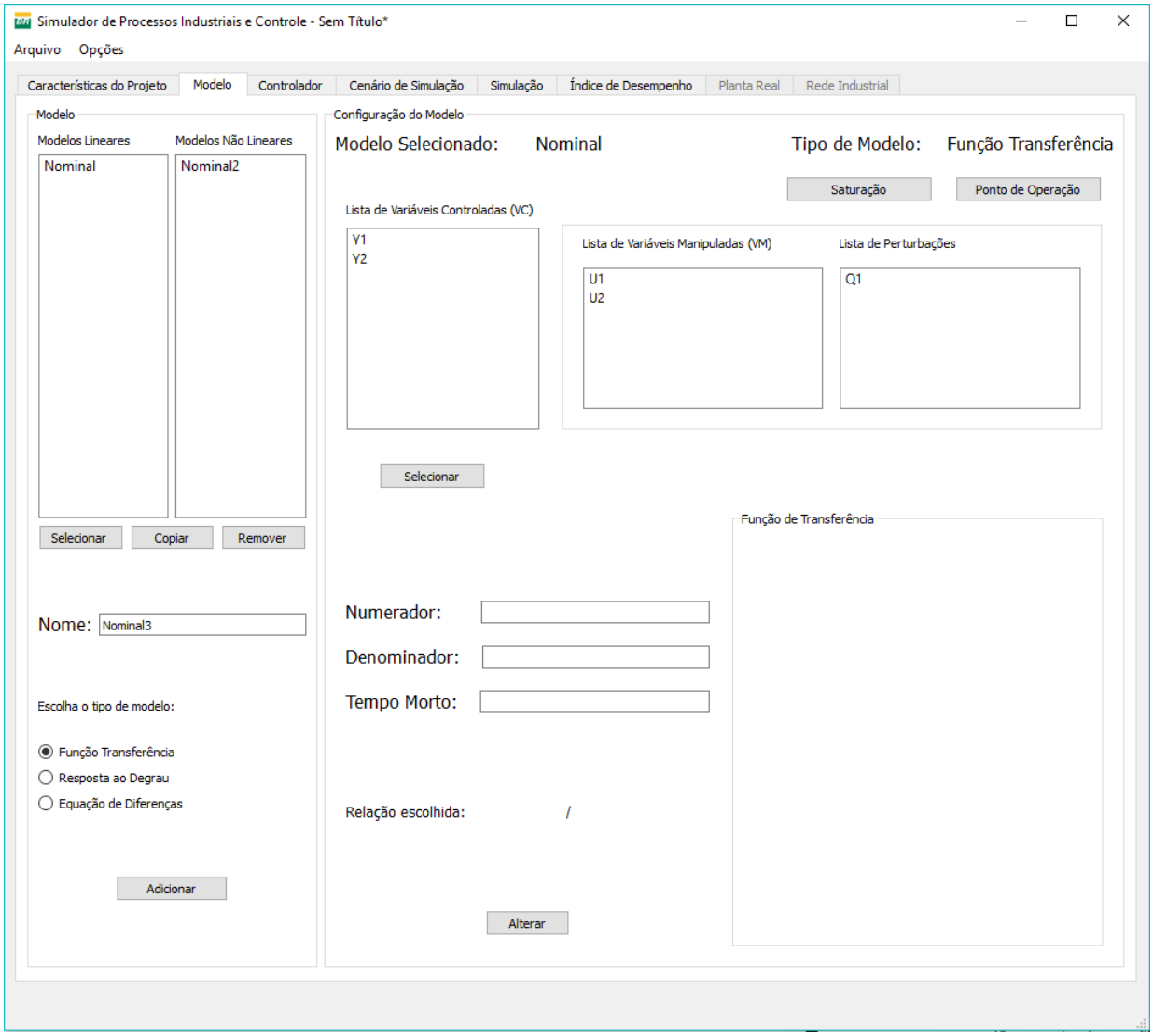

Figura 30 – Aba dos modelos.

Foram implementados três tipos de modelos para representar os processos: *Função Transferência Discreta*, *Resposta ao Degrau* e *Equações de Diferença*. Os dois primeiros foram definidos para sistemas lineares discretos, enquanto o último representa sistemas não-lineares e, dessa forma, foram criadas duas listas para diferenciar os modelos lineares e não-lineares na interface, como pode ser visto na Figura 31. Por questão de comodidade ao usuário, quando o projeto foi iniciado o sistema automaticamente já adiciona dois modelos na lista, um modelo linear do tipo função transferência e outro não linear (mostrado na figura com nomes *Nominal* e *Nominal2*).

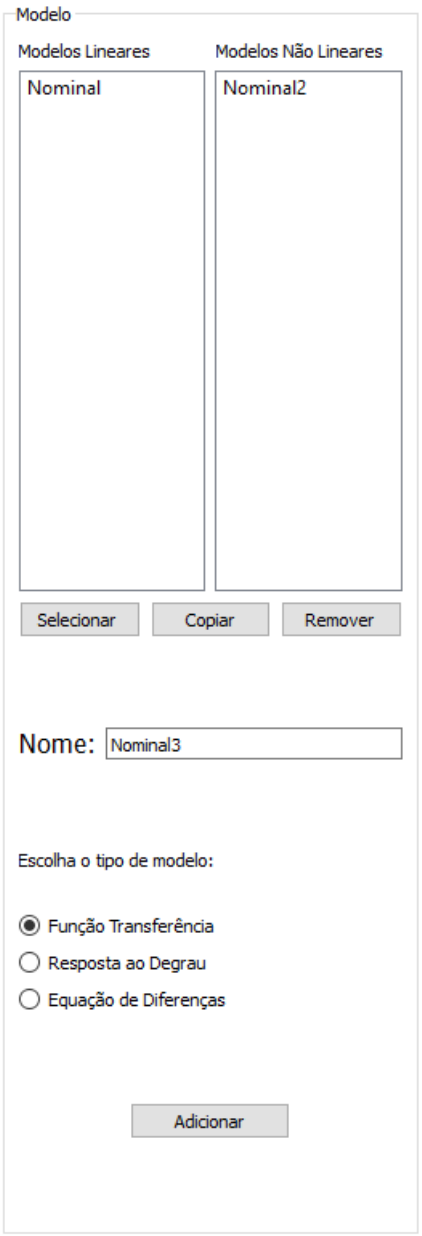

Figura 31 – Lista de modelos.

Para criar um novo modelo o usuário primeiramente define o seu *Nome* e define o tipo do modelo, clicando no botão *Adicionar*. Existem algumas restrições quanto ao

nome do novo modelo que se executadas elas geram uma janela semelhante a da Figura 27:

- Não podem ser criados modelos com nomes já existentes nas listas;
- Os nomes devem ser de no máximo 12 caracteres.

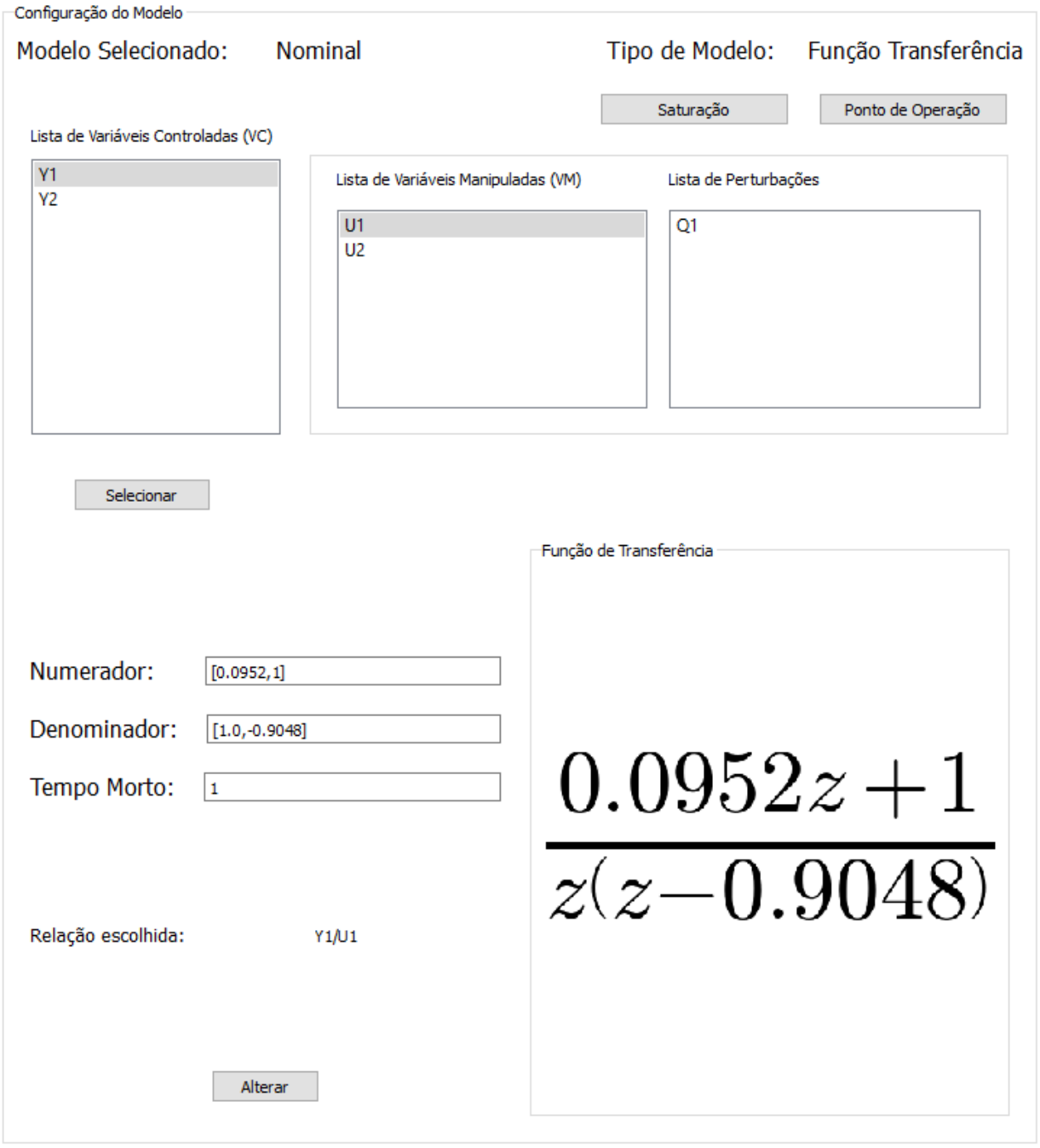

Figura 32 – Exemplo de modelo tipo função transferência selecionado.

A Figura 31 também revela alguns botões adicionais logo abaixo das listas: *Selecionar*, *Copiar* e *Remover*. O primeiro serve para o usuário definir um modelo para

modificar suas variáveis e configurar seus parâmetros, de forma que o usuário **sempre** deve clicar num dos modelos da lista para selecioná-los. O mesmo também acontece com o botão Copiar que ao ser clicado com um item da lista selecionado abre uma janela e pergunta ao usuário se ele deseja copiar o modelo para a lista e qual o nome do novo modelo. Por fim, o botão Remover retira um dos itens selecionados da lista quando clicado.

Como citado anteriormente, a própria interface se adapta ao tipo de modelo selecionado, porém alguns elementos da interface são comuns a todos os tipos (ver Figura 32). Esses elementos são:

- Saturação do sinal de controle;
- Ponto de operação das variáveis;
- Lista de variáveis controladas, manipuladas e perturbações.

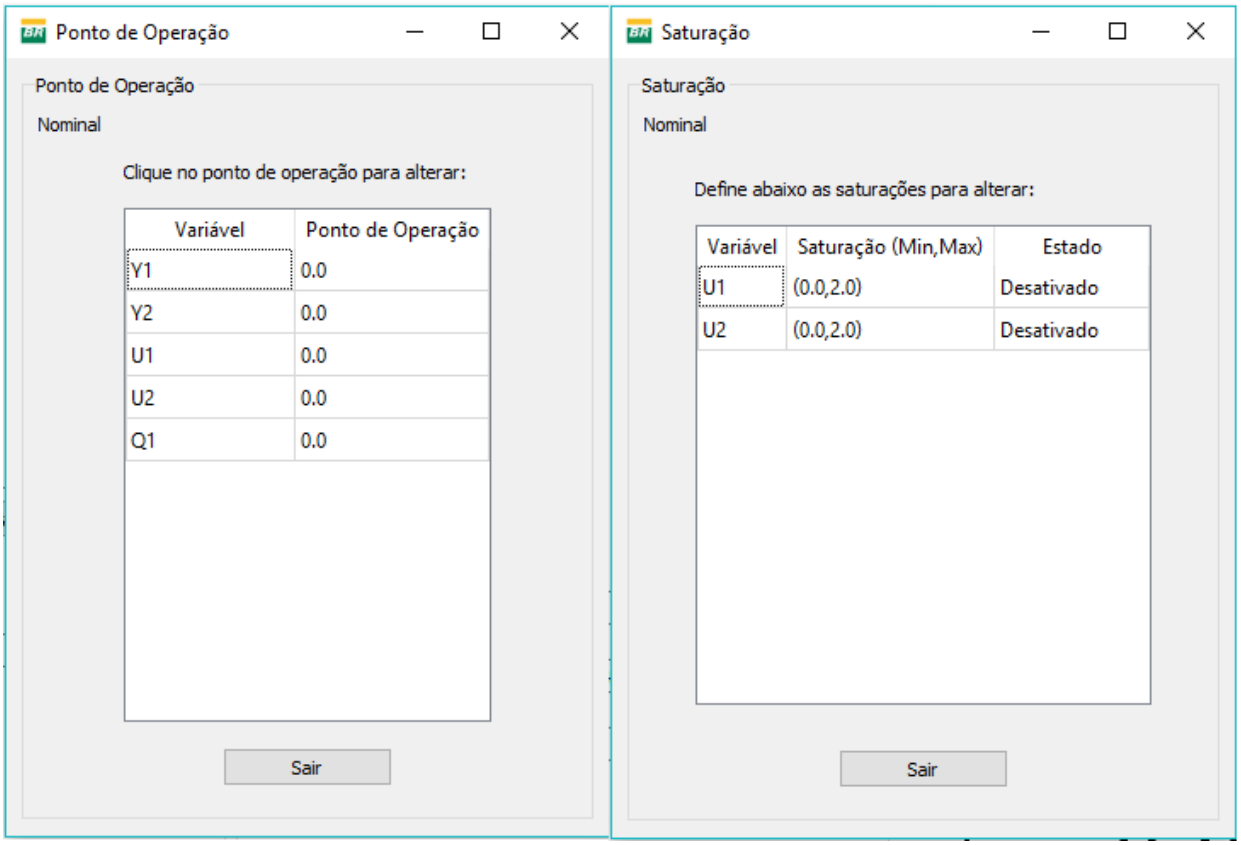

Figura 33 – Janelas para alterar a saturação e os pontos de operação.

A saturação do sinal de controle e os pontos de operações foram implementados criando botões na interface, na parte superior direita da aba, de maneira que quando clicadas ambas abrem janelas (iguais a Figura 33) semelhantes que permitem a configuração desses parâmetros. Os dois são similares no sentido de que apresentam

uma tabela com as variáveis na primeira coluna e os parâmetros de configuração nas colunas seguintes. No caso dos pontos de operação o usuário pode definir valores do tipo *float* para qualquer variável do sistema. Já para saturação, apenas as variáveis manipuladas apresentam saturações e o usuário define um valor mínimo e máximo (ou seja, um intervalo de saturação) e também define se a saturação está 'Ativada' ou 'Desativada', que indica se durante a simulação as entradas realmente irão apresentar um intervalo de saturação.

No exemplo da Figura 32 foi selecionado um modelo do tipo função transferência, cujo nome é *Nominal*. Cada tipo de modelo tem suas características próprias que podem ser ajustadas pelo usuário e para isso ele deve escolher quais as variáveis do processo que serão modificadas. No exemplo da Figura 32 o usuário escolheu um par entrada e saída específico da lista de variáveis manipuladas e controladas para modificar a função transferência do modelo.

Modelo Função Transferência

Como já citado, para modelos do tipo função transferência o usuário deve selecionar um item da lista de variáveis controladas e outro da lista de variáveis manipuladas ou perturbações. É necessário escolher um par de variáveis pois na representação por funções transferência (ver Equação 2.6) cada saída é relacionada com uma entrada ou perturbação, criando funções definidas por numeradores, denominadores e o tempo morto do sistema. Para configurar esses parâmetros é utilizado um formato específico para representá-los:

- Sempre definidos entre colchetes;
- Os números são separados por vírgulas e utiliza-se ponto para separar as casas decimais;
- Pode-se entender como uma lista $L = [k_1, k_2, \ldots, k_n]$ de tamanho n, onde cada elemento da lista representa um fator que multiplica um componente do polinômio $P = [z^{n-1}, z^{n-2}, \ldots, z^0]$ elemento por elemento. Por exemplo, na Figura 32 o numerador é uma lista $L = [0.0952, 1.0]$, com n igual a 2, que então multiplica o polinômio $P = [z^1, z^0]$ elemento por elemento, resultando em:

$$L. * P = [0.0952 * z^1, 1.0 * z^0] \tag{5.1}$$

  onde o símbolo .* representa a multiplicação elemento por elemento;
- O tempo morto é definido apenas por um número inteiro.

Para alterar o numerador, denominador e tempo morto do par escolhido, o usuário só precisa clicar no botão *Alterar*.

Modelo Resposta ao Degrau

Assim como no caso anterior, os modelo do tipo resposta ao degrau exigem que o usuário selecione um par entrada/saída ou perturbação/saída (ver Equação 2.1). Mas nesse caso, os parâmetros de configuração são diferentes, já que a estrutura do modelo é outra. Na Figura 34 pode-se constatar que o usuário configura dois parâmetros distintos, um vetor de coeficientes g_i, definido entre colchetes e separando cada elemento por vírgulas, e o número de coeficientes do vetor g_i. É importante que durante a definição do vetor g_i o tamanho do vetor coincida com o número determinado na interface, caso contrário um aviso será gerado informando que a ação foi inválida.

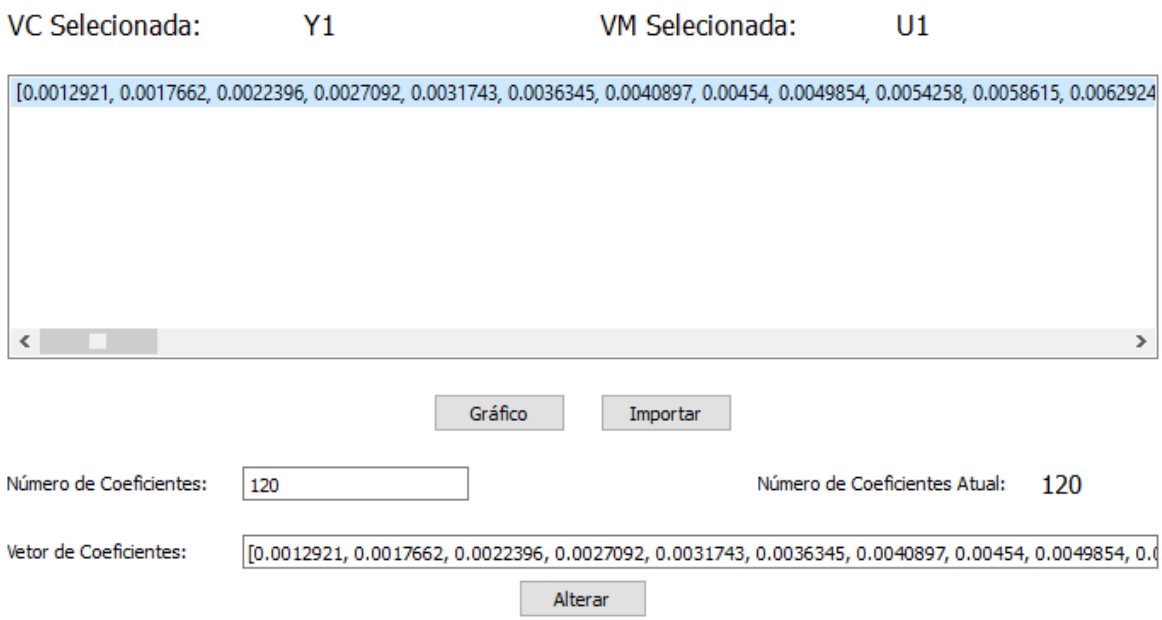

Figura 34 – Seção da interface do modelo resposta ao degrau.

Ao clicar no botão *Gráfico*, o sistema gera um gráfico que mostra o vetor g_i, num intervalo que começa em zero e termina no número de coeficiente definido pelo usuário.

Outra ferramenta implementada diz respeito ao botão *Importar*, que permite ao usuário importar arquivos externos para a interface que contêm vetores de coeficientes já previamente criados num formato específico, cuja a extensão é o .dmi. O principal objetivo da implementação dessa ferramenta foi auxiliar projetos já existentes em parceria com o centro de pesquisa da Petrobrás, reduzindo o tempo necessário para importar os dados do modelo direto para a interface.

Por fim, semelhante ao modelo função transferência, o usuário altera os parâmetros clicando no botão *Alterar*.

Modelo Equação de Diferenças

Diferentemente dos casos apresentados anteriormente, o modelo de equação de diferenças requer apenas a seleção de uma saída, visto que uma mesma variável controlada pode apresentar na sua equação qualquer variável do sistema e portanto o modelo não representado por pares (ver Equação 2.7).

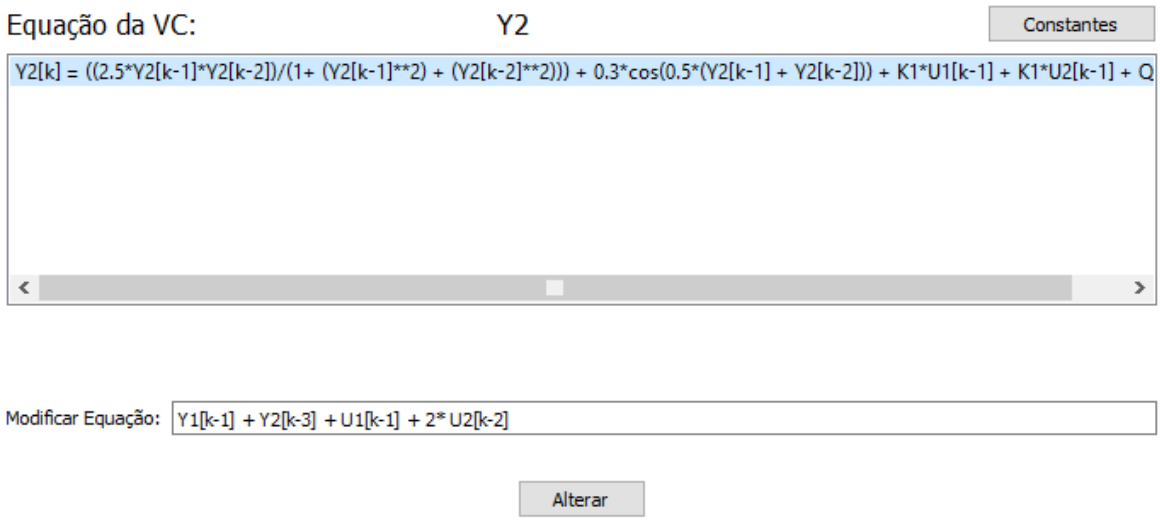

Figura 35 – Seção da interface do modelo de equação de diferenças.

A equação é escrita intuitivamente, conforme a Figura 35, com formato conhecido em diferentes softwares computacionais que apresentam equações matemáticas, facilitando a utilização para usuários familiarizados com softwares como Matlab ou Octave (no caso do modelo linear também foi utilizado padrões difundidos em softwares de desenvolvimento de algoritmos matemáticos). A equação deve ser indicada sempre com a variável k para representar os instantes de tempo, entre colchetes e o nome das variáveis do processo devem ser exatamente iguais aos definidos na aba *Características do Projeto*.

Os modelos não-lineares apresentam uma característica única que é a possibilidade de definir constantes do processo (Figura 36). Isso permite ao usuário criar equações que dependem de certas constantes e simular o sistema modificando as constantes em diferentes cenário. Por exemplo, pode-se iniciar um projeto de controle de nível onde o tanque pode apresentar diferentes volumes. Pode-se então criar uma constante chamada *Volume* e atribuir valores distintos, como 100 litros ou 300 litros, e depois criar diferentes cenário que simulam o comportamento do sistema para cada volume.

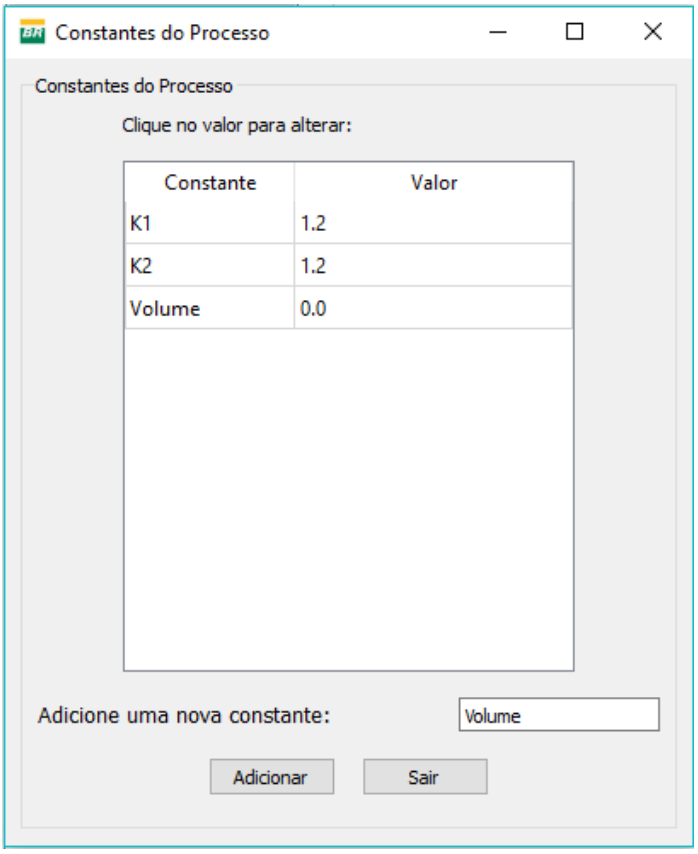

Figura 36 – Janela para definição de constantes.

5.3 Controlador

No Capítulo 2 foram apresentados os algoritmos de controle preditivo MPC implementados na interface, foco principal do software. Portanto, a aba dos *Controladores* foi criada para permitir a definição de diferentes tipos de controladores utilizando a mesma ideia dos aba do modelos que se adaptava para os diferentes tipos escolhidos. O resultado é apresentado na Figura 37 que mostra alguns dos elementos da aba para definição dos controladores, bastante semelhantes à aba modelo.

Os controladores podem ser do tipo *GPC*, *PNMPC*, *DMC* e *PID*. Cada tipo de controlador tem atributos únicos, visto que seus algoritmos são diferentes entre si, porém no caso dos algoritmos preditivos há atributos semelhantes que são explorados. Os controladores PID foram implementados principalmente para realização de testes da performance do MPC, pois são controladores amplamente estudados e com resultados mais previsíveis, portanto são bons para comparar com simulações utilizando algoritmos preditivos.

A criação dos controladores é realizada de maneira semelhante à aba modelo, na Figura 38 pode-se perceber que a própria estrutura é similar. O objetivo é manter a coerência dos elementos da interface, na tentativa de padronizar esses elementos para

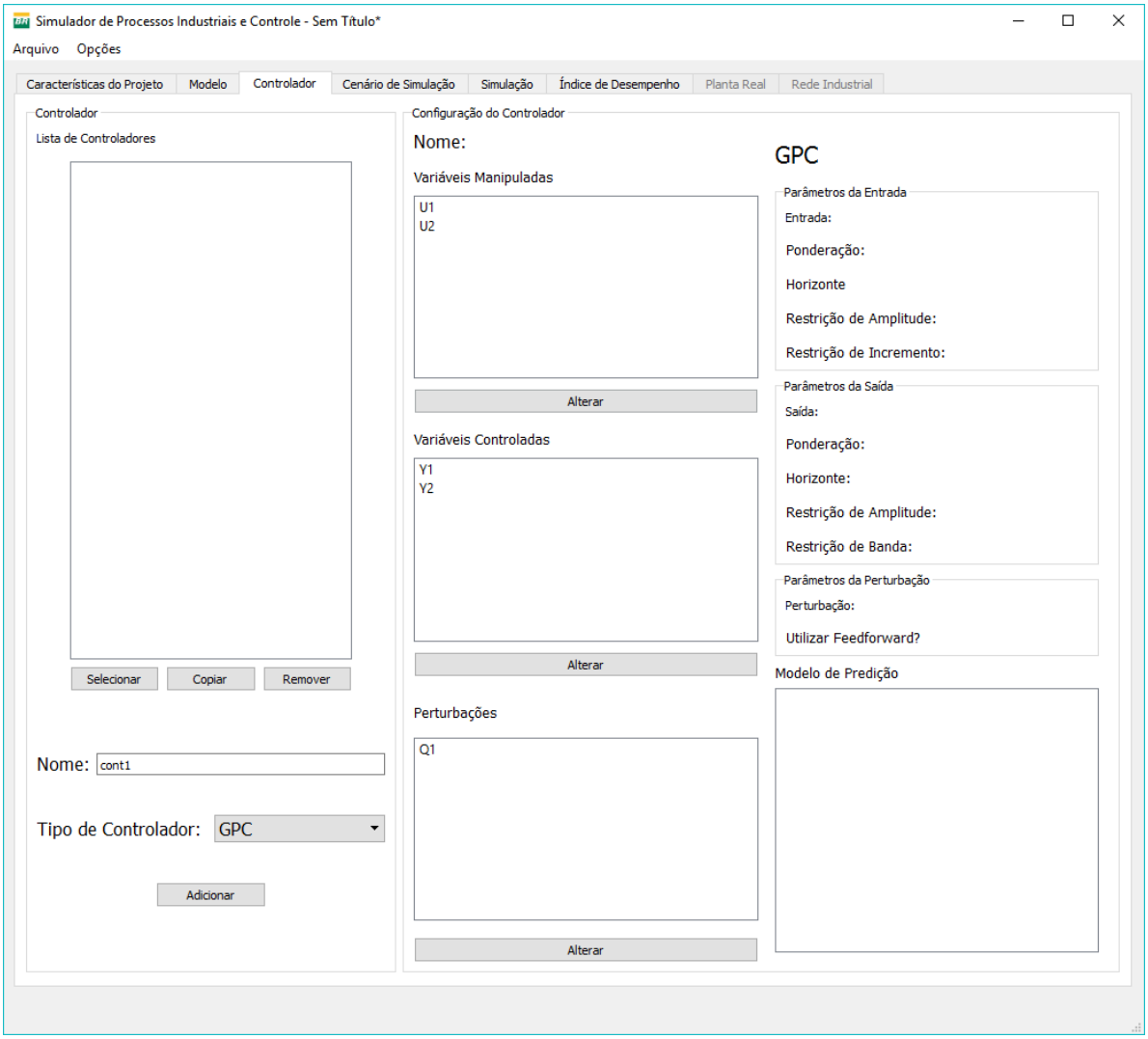

Figura 37 – Aba dos controladores.

auxiliar na utilização do software. Resumindo a definição dos controladores, o usuário escolhe um nome para o controlador, com as mesmas restrições dadas aos modelos e também não permitindo a criação de controladores com nome igual aos modelos criados anteriormente. Em sequência, o usuário escolhe o tipo do controlador e clica no botão *Adicionar*. Na Figura 38 foram adicionados três controladores à lista, com nomes *cont1*, *cont3* e *cont2*, como exemplo.

Na aba também foram adicionados três botões logo abaixo da lista *Selecionar*, *Copiar* e *Remover*, com as mesmas funções dos adicionados para os modelos. Todos exigem que um item da lista de controladores tenha sido clicado, gerando uma janela de aviso caso o usuário clique no botão sem um item selecionado.

Ao selecionar um controlador, a interface irá se adaptar ao tipo escolhido pelo usuário e, com isso, o usuário pode configurar os parâmetros do controlador que serão utilizados durante a simulação para o cálculo das ações de controle. Na Figura

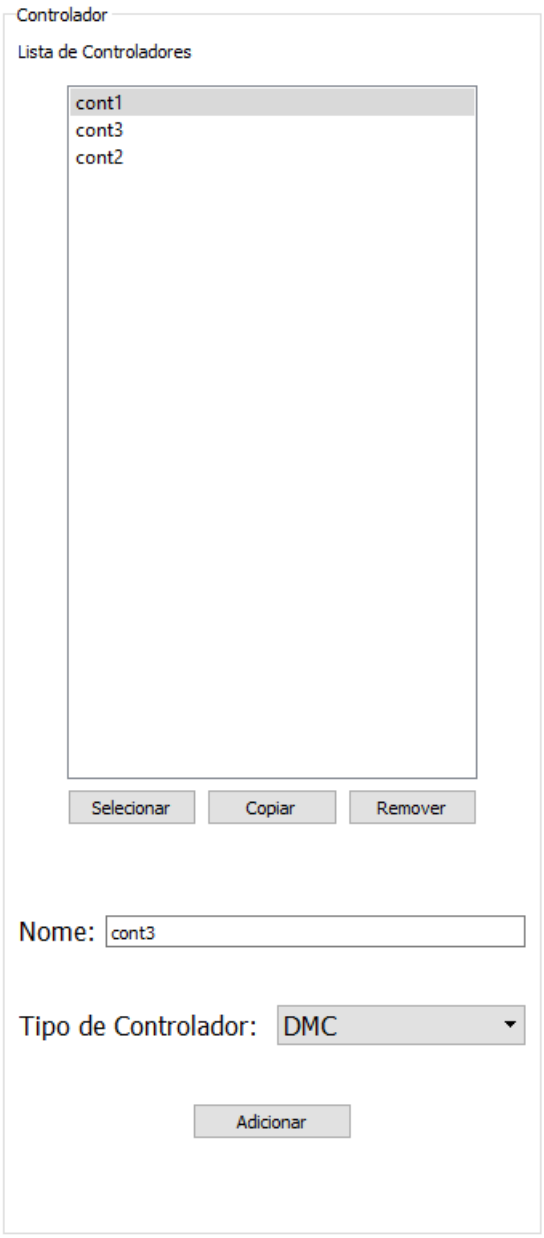

Figura 38 – Definição dos controladores.

39 é mostrada a seção da janela responsável pela configuração dos parâmetros do controlador, composta por três listas para cada variável do processo no lado esquerdo e as informações de cada variável no lado direito. Essas informações são atualizadas automaticamente sempre que o usuário clica numa das variáveis (sem pressionar no botão *Alterar*). Além disso, quando selecionado um controlador preditivo irá aparecer no canto inferior direito uma lista que contém os modelos, permitindo ao usuário selecionar qual será o modelo de predição do controlador clicando num dos itens da lista.

A configuração dos parâmetros se dá clicando no botão *Alterar* posicionado abaixo das listas de variáveis do processo, após o usuário escolher um item das listas, abrindo um janela para a configuração. Por exemplo, na Figura 39 foi selecionado o

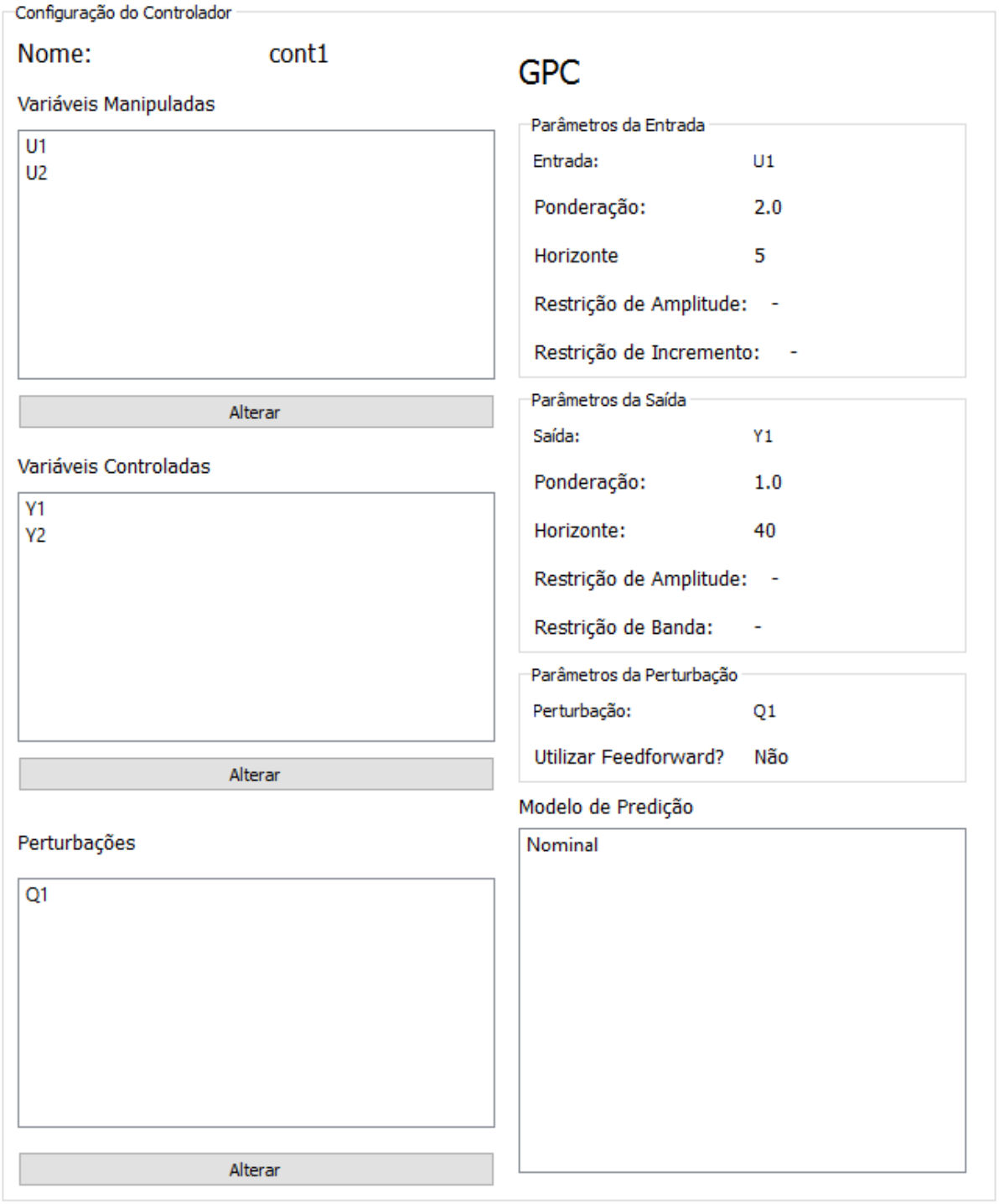

Figura 39 – Configuração dos controladores.

controlador *cont1* e o usuário deseja alterar um parâmetro da variável manipulada *U1*. Para isso ele deve selecioná-la na lista de variáveis manipuladas e então clicar no botão *Alterar*, o que resultando na aparição da janela mostrada na Figura 40.

Para os algoritmos preditivos, existem parâmetros comuns aos três tipos definidos, simplificando a implementação da lógica de programação (utilizando o conceito de classes). Esses parâmetros são:

- Variáveis Manipuladas;
  - Ponderação;
  - Horizonte de entrada;
  - Restrição de Amplitude;
  - Restrição de Incremento de controle.
- Variáveis Controladas;
  - Ponderação;
  - Horizonte de saída;
  - Restrição de Amplitude;
  - Restrição de Banda.
- Perturbações.
  - Ação Feedforward.

As restrições para cada parâmetro são:

- Ponderações devem sempre ser positivas;
- Horizontes devem assumir valores inteiros positivos;
- A amplitude máxima das restrições deve sempre ser maior que a mínima;
- O usuário pode desativar as restrições.

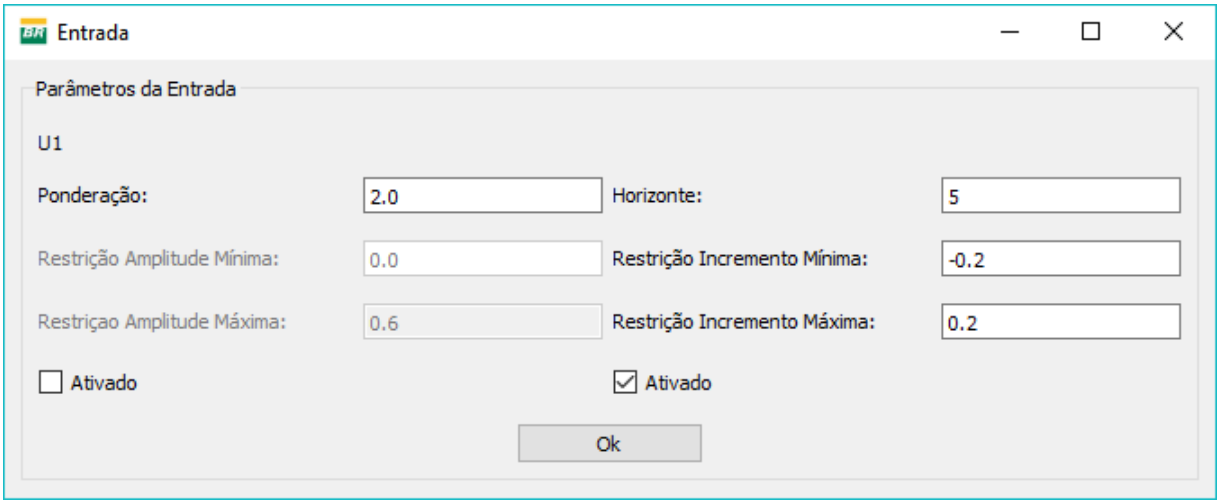

Figura 40 – Janela para alteração dos parâmetros de entrada do controlador.

O que diferencia cada algoritmo preditivo é como são calculadas as predições do processo. Os algoritmos apresentam modelos diferentes para o cálculo, como explicado com detalhes no Capítulo 2, influenciando nos modelos que são mostrados na lista de modelos de predição. Portanto, na lista somente serão mostrados ao usuário os modelos que são permitidos utilizar para calcular a predição dependendo do tipo do controlador. Por exemplo, na Figura 39 apenas o modelo *Nominal* está listado, pois o controlador selecionado é do tipo GPC e o modelo *Nominal* é do tipo função transferência.

5.4 Cenário

Todos os elementos componentes básicos para que o sistema execute a simulação do processo já foram criados. Agora, seguindo a sequência descrita anteriormente, o usuário deve configurar a simulação antes de executá-lo. O principal objetivo dessa configuração é recriar situações reais, buscando representar processos industriais da forma mais fiel possível e gerando cenários de simulação.

Portanto, a aba de *Cenários* deve ser responsável pela configuração da simulação. Na Figura 41 é mostrado o resultado da implementação, que assim como nas duas abas anteriores, exibe uma lista no lado esquerdo da interface, responsável por armazenar todos os cenário criados. Para criar um cenário, o usuário escolhe um nome, atendendo todas as restrições e então clica no botão *Adicionar*. Também foram adicionados os mesmos botões *Selecionar*, *Copiar* e *Remover* que funcionam conforme descrito anteriormente.

Cada cenário tem características próprias, que são modificados na seção da janela mostrado na Figura 42. As principais características são:

- Tempo de simulação ou número de iterações;
- Estado inicial;
- Tipo do cenário (malha aberta ou malha fechada);
- Sinais de referência (caso malha fechada), entrada (caso malha aberta) e perturbação;
- Modelo de simulação;
- Controlador.

Cada uma dessas variáveis tem papel fundamental na configuração do cenário. O tempo de simulação determina a duração necessária para finalizar toda a simulação

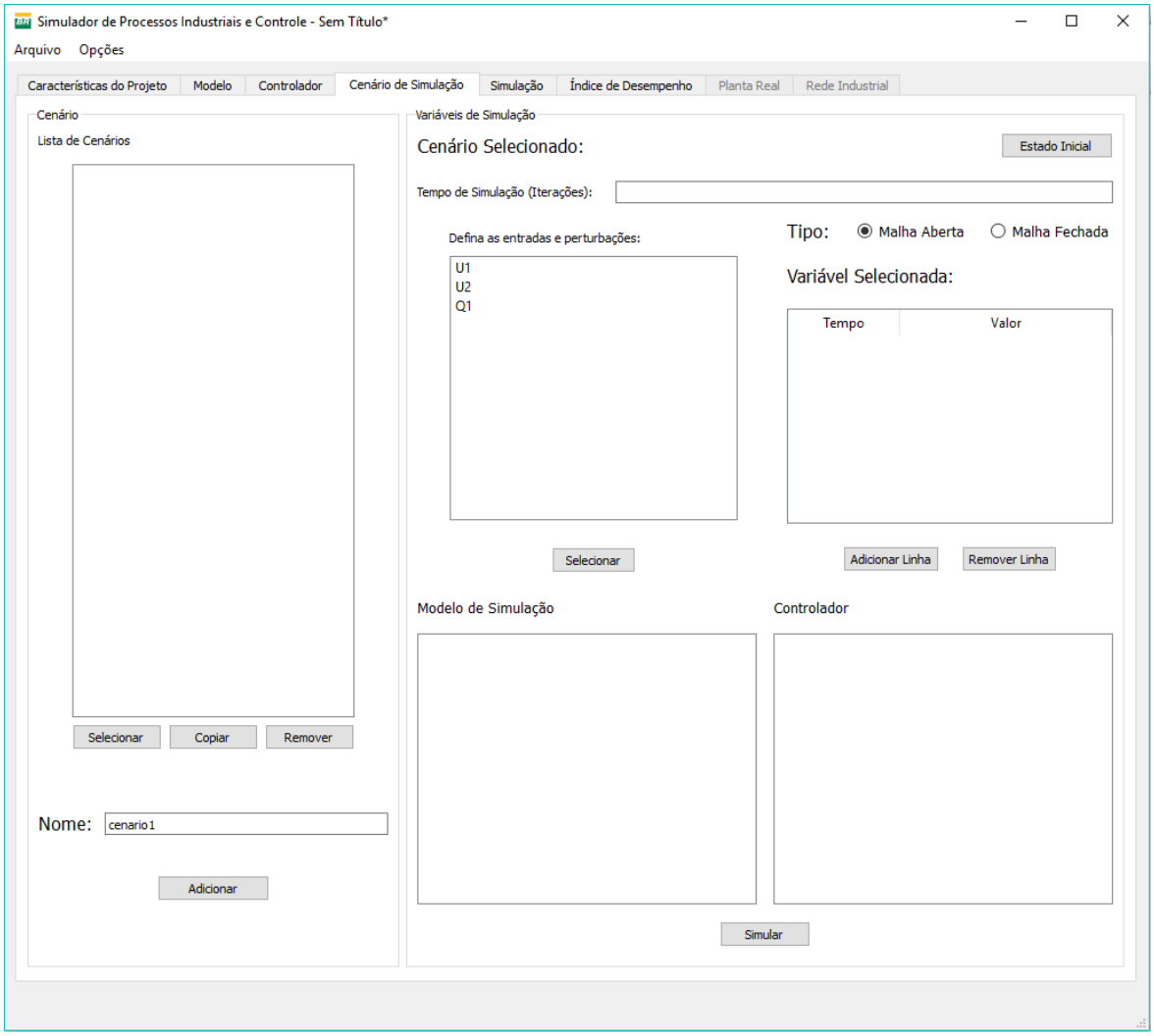

Figura 41 – Aba dos cenários.

em *iterações*. O *estado inicial* determina quais as condições que cada variável do processo se encontra no início da simulação, alterando os valores clicando no botando posicionado no canto superior direito da tela. O *tipo* de cenário interfere na simulação, definindo se o sistema será realimentado ou não. Quando o tipo é escolhido, o usuário pode definir as referências (ou entradas) e perturbações do sistema, atribuindo valores em diferentes instantes de tempo, numa tabela posicionada no centro direito da tela. Esses valores atribuídos podem ser definidos pelo usuário como sinais específicos:

- Para aplicar um sinal do tipo **degrau** o usuário define apenas um valor para a variável, como 1, 5 ou 10;

- Para aplicar um sinal do tipo **rampa**, deve-se na coluna *Valor* da tabela digitar *k*, *2k*, *10k*, etc. A constante *k* representa o instante incremental de tempo e o fator que o multiplica apenas indica a inclinação da rampa;

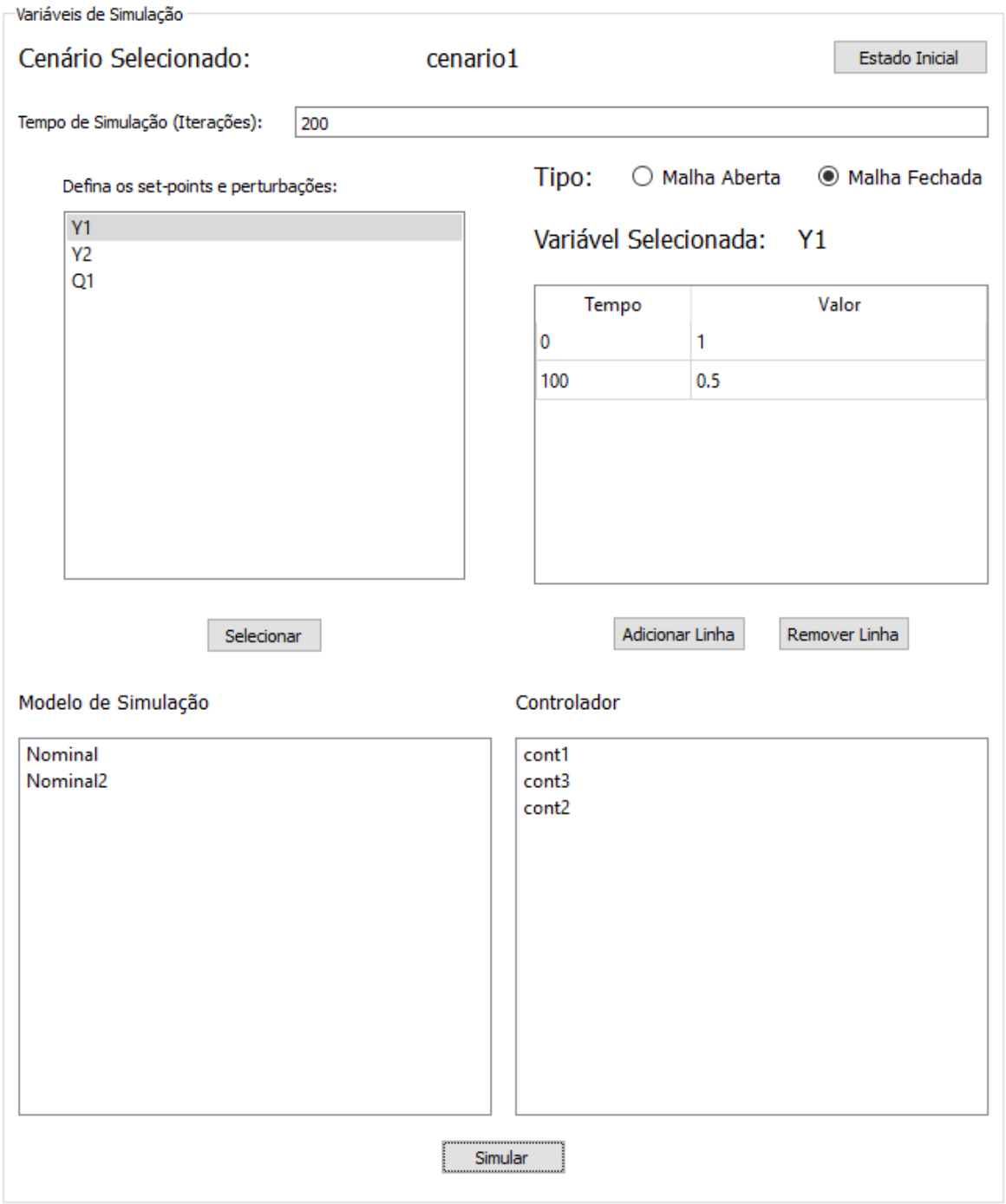

Figura 42 – Janela para configuração dos cenários de simulação.

- Para aplicar **senóides** de entrada basta no campo *Valor* da tabela o usuário digitar *sen(k)*, *sen(60k)*, *cos(k)*, *cos(30)*, etc;
- Por fim, o usuário pode definir sinais **aleatórios**, digitando *random(a,b)*. As constantes 'a' e 'b' podem assumir qualquer valor, desde que $a < b$. O resultado é um valor aleatório entre o intervalo $[a, b]$.

Clicando no botão *Adicionar Linhas* pode-se criar novas referências, sinais de

entrada ou perturbações (ver Figura 43). Portanto, o usuário pode definir a quantidade necessária de sinais para cada variável, incluindo diferentes sinais de referência para cada saída por exemplo. Analogamente, pode-se remover um sinal apertando no botão *Remover Linha*.

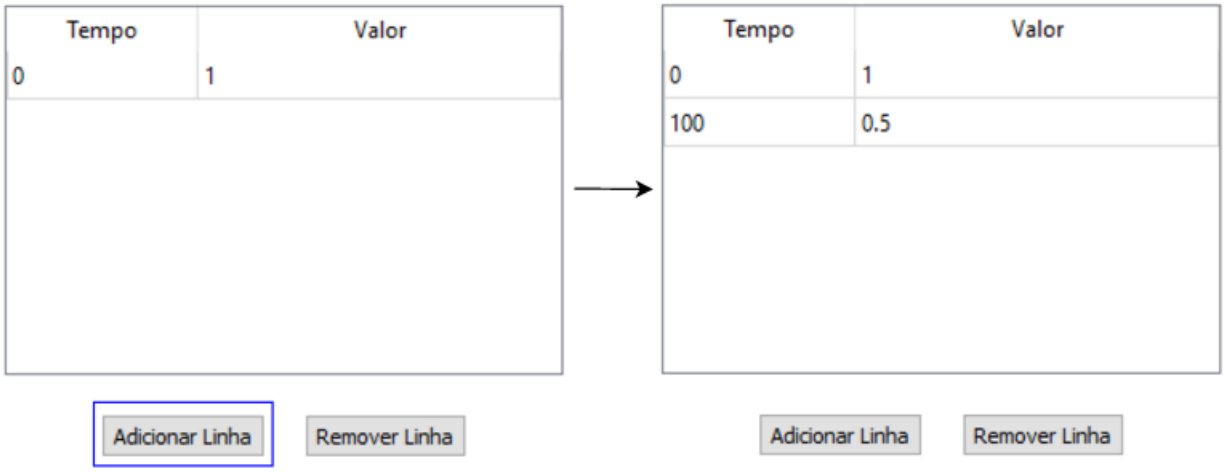

Figura 43 – Demonstração de como modificar a referência.

Por fim, o usuário pode simular o cenário já configurado, selecionando um modelo, um controlador e clicando no botão *Simular*. Para que a simulação tenha sucesso, o usuário também deve escolher um modelo de predição, na aba do controlador, para o controlador escolhido. Com todas as pré-condições da simulação satisfeitas, todos os cálculos são realizados, determinando as ações de controle para que o sistema alcance as referências selecionadas. Durante o cálculo aparece um aviso ao usuário para aguardar até que a simulação seja finalizada no canto inferior esquerdo, na parte cinza da tela e, após o término da simulação, uma janela é gerada para informar ao usuário que a operação foi realizada com sucesso.

5.5 Simulação

Nesse momento o sistema já fez todos os cálculos necessários para o usuário analisar a simulação. Portanto, na aba *Simulação* (Figura 44), como o próprio nome revela, são recuperados os dados calculados anteriormente e são fornecidos gráficos para o usuário interpretá-los. Uma das suas principais características é que pode-se produzir gráficos de diferentes cenários ao mesmo tempo, ou seja, o usuário pode criar diferentes situações que envolvam o modelo, como controladores diferentes ou parâmetros de entrada/referências diferentes e simular ambos cenários para comparar suas performances, conforme a sua necessidade. Outra informação importante é que o usuário pode escolher as variáveis que ele deseja observar nos gráficos.

Portanto, para gerar um gráfico da simulação o usuário deve:

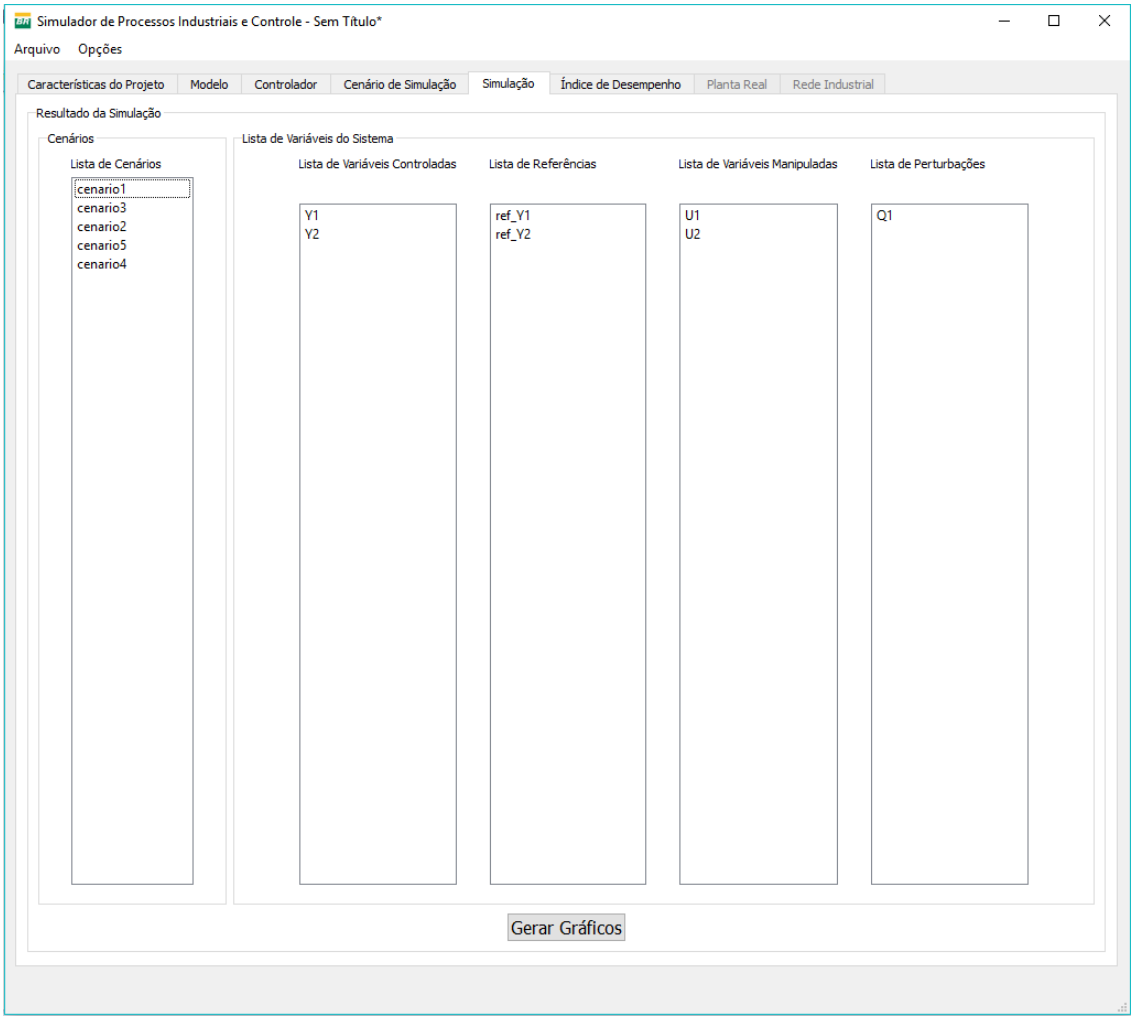

Figura 44 – Aba de simulação.

1. Selecionar os cenários desejados na Lista de Cenários;

2. Selecionar as variáveis do processo que deseja-se analisar nas suas respectivas listas;

3. Clicar no botão *Gerar Gráficos*.

Um exemplo pode ser observado na Figura 45. Nessa simulação foi definido um sistema com duas entradas (U_1 e U_2), duas saídas (Y_1 e Y_2) e uma perturbação (Q_1) e cenário cujo controlador era do tipo GPC, com $N_u = 5$, $N_y = 40$, $Q_u = 2.0$, $Q_y = 1.0$ e sem restrições. A função transferência do modelo é desacoplada, com:

$$\frac{Y}{U} = \frac{0.0952}{z - 0.9048} \qquad \frac{Y}{Q} = \frac{0.0952}{z(z - 0.9048)} \tag{5.2}$$

O cenário foi configurado da seguinte forma:

- Número de iterações igual a 200;
- Tipo do cenário é malha fechada;
- Referências e perturbações em:
  - em $k = 0 \rightarrow$ referência $= 1.0$ para as duas saídas;
  - em $k = 50 \rightarrow$ perturbação $Q_1 = 0.5$;
  - em $k = 100 \rightarrow Y_1$ com referência $= 0.5$ e Y_2 com referência $= 1.5$.

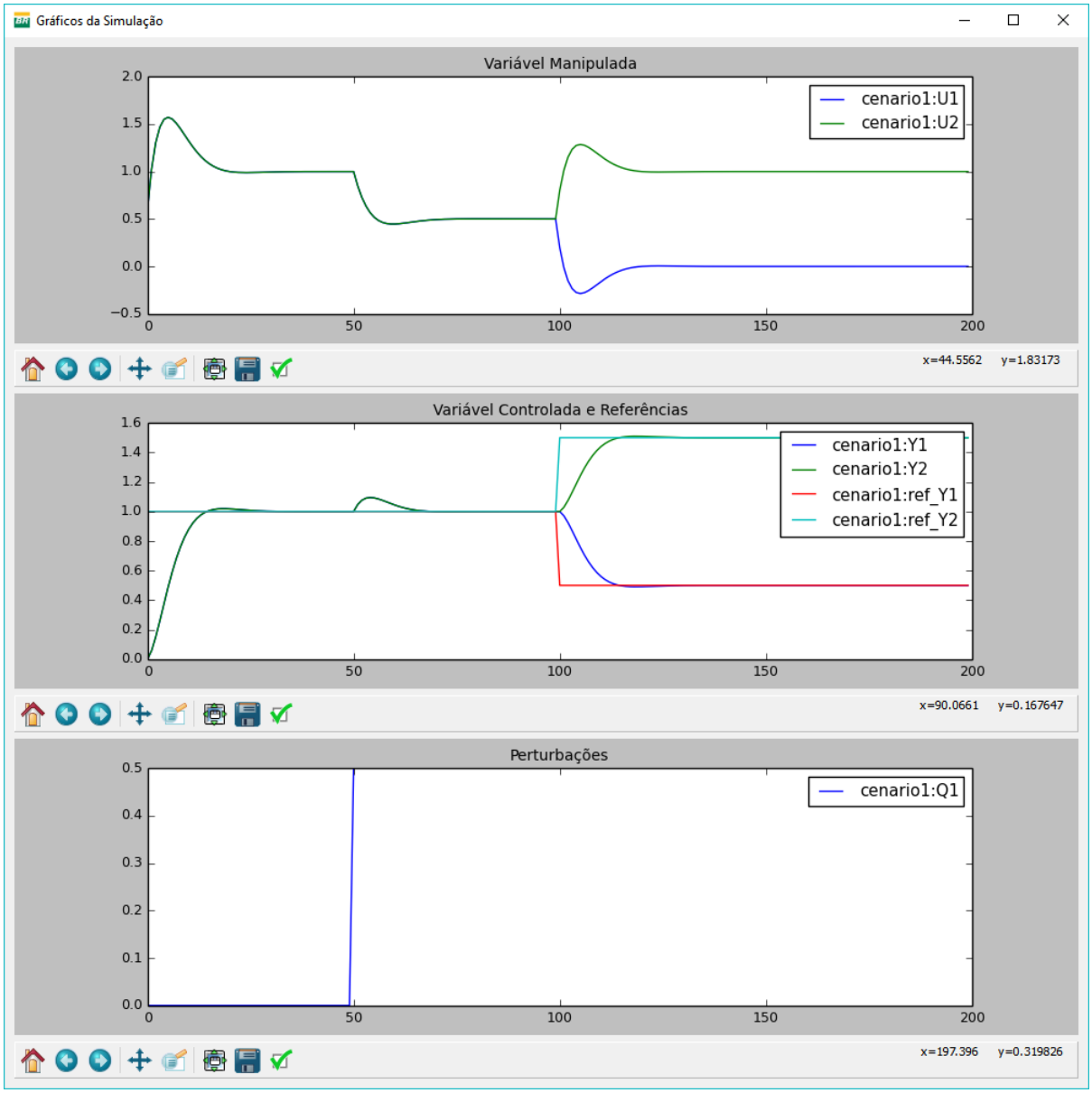

Figura 45 – Exemplo de gráfico gerado.

Também foi criado um exemplo de simulação para um sistema não-linear, cuja equação de diferença é dada por

$$\begin{aligned} Y_i[k] = & \frac{2.5Y_i[k-1]Y_i[k-2]}{1+Y_i[k-1]^2} + Y_i[k-2]^2 \\ & + 0.3\cos 0.5(Y_i[k-1] + Y_i[k-2]) \\ & + K_1U_1[k-1] + K_1U_2[k-1] + Q_1[k-1] \end{aligned} \tag{5.3}$$

sabendo que $i = 1, 2$, K_1 e K_2 são duas constantes definidas na aba dos modelos e que $K_1 = K_2 = 1.2$.

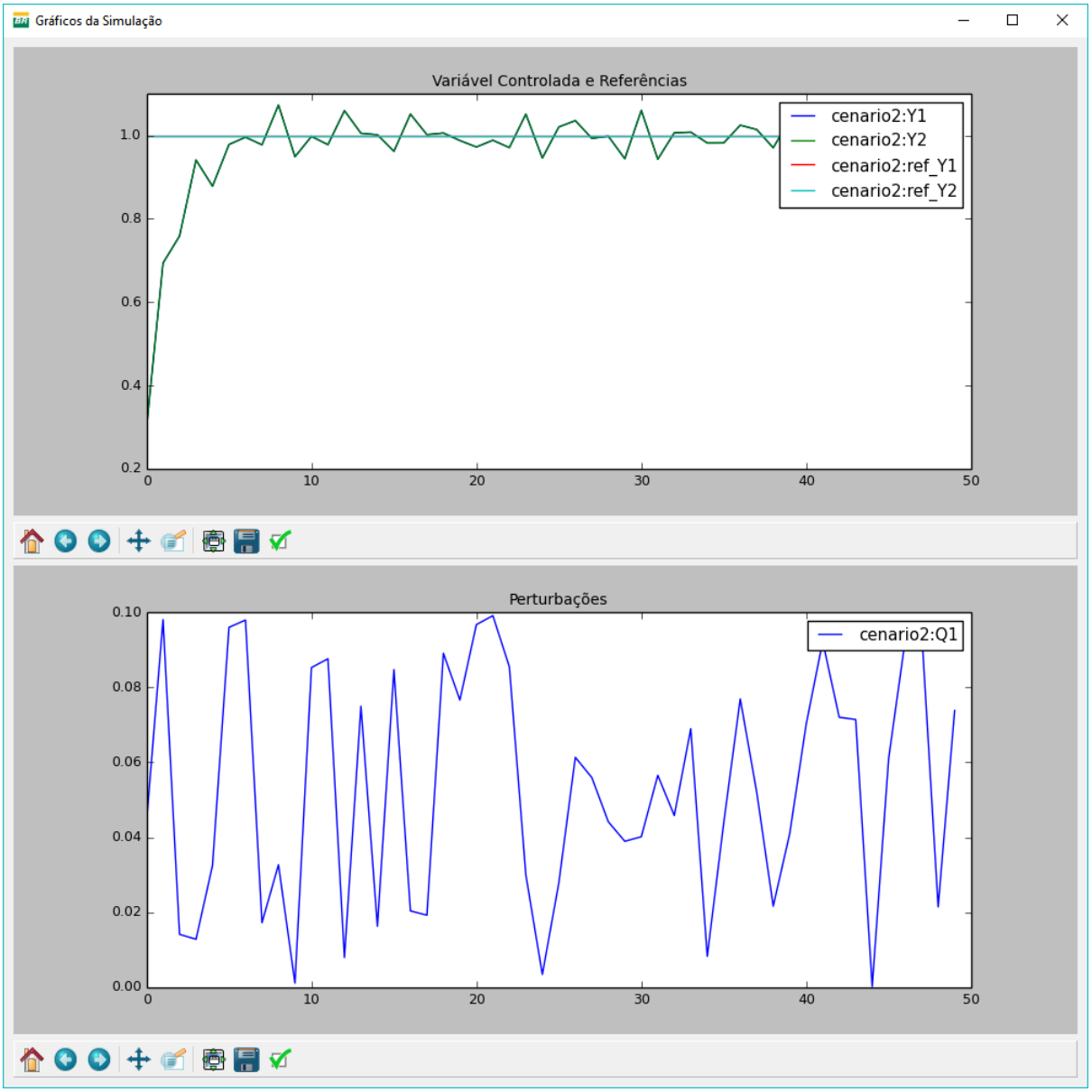

Figura 46 – Outro exemplo de gráfico gerado.

O controlador definido é do tipo PNMPC, com $N_u = 5$, $N_y = 40$, $Q_u = 2.0$, $Q_y = 1.0$ e sem restrições. O cenário foi configurado da seguinte forma:

- Número de iterações igual a 50;
- Tipo do cenário é malha fechada;
- Referências e perturbações em:
  - em $k = 0 \rightarrow$ referência $= 1.0$ para as duas saídas;
  - em $k = 0 \rightarrow$ perturbação $Q_1 = random(0, 0.01)$.

O resultado é mostrado na Figura 46. Observa-se que foram selecionados apenas as variáveis controladas e as perturbações nas lista da aba simulação, de forma que apenas dois gráficos foram gerados.

5.6 Índice de Desempenho

Os índices de desempenho são necessários para analisar a robustez do controlador criado no cenário de simulação definido pelo usuário. Ferramentas de *avaliação* da performance do MPC dão subsídios ao engenheiro para melhorar a *sintonia* dos controladores ou até mesmo para indicar quando um modelo de predição é insatisfatório [37]. Os algoritmos implementados para calcular os índices originaram-se durante o mestrado de Paulo Cortez na Universidade Federal de Santa Catarina.

Foram implementados os seguintes índices de desempenho:

- Índices de Avaliação
  - Desvios Médios Individuais;
  - Índices de Estabilidade;
  - Desvio Médio Total.
- Índices de Sintonia
  - Índices de Seguimento Relativo;
  - Índices de Erro de Modelagem;
  - Índices de Controle Relativos;
  - Índices de Supressão de Movimento.

Portanto, o resultado foi a aba dos *Índices de Desempenho* proposta na Figura 47. Também foi implementada uma variável chamada de *Fator de Esquecimento*, que permite uma avaliação contínua do resultado possibilitando a atualização dos índices [37] em sistemas com tempo de simulação longos.

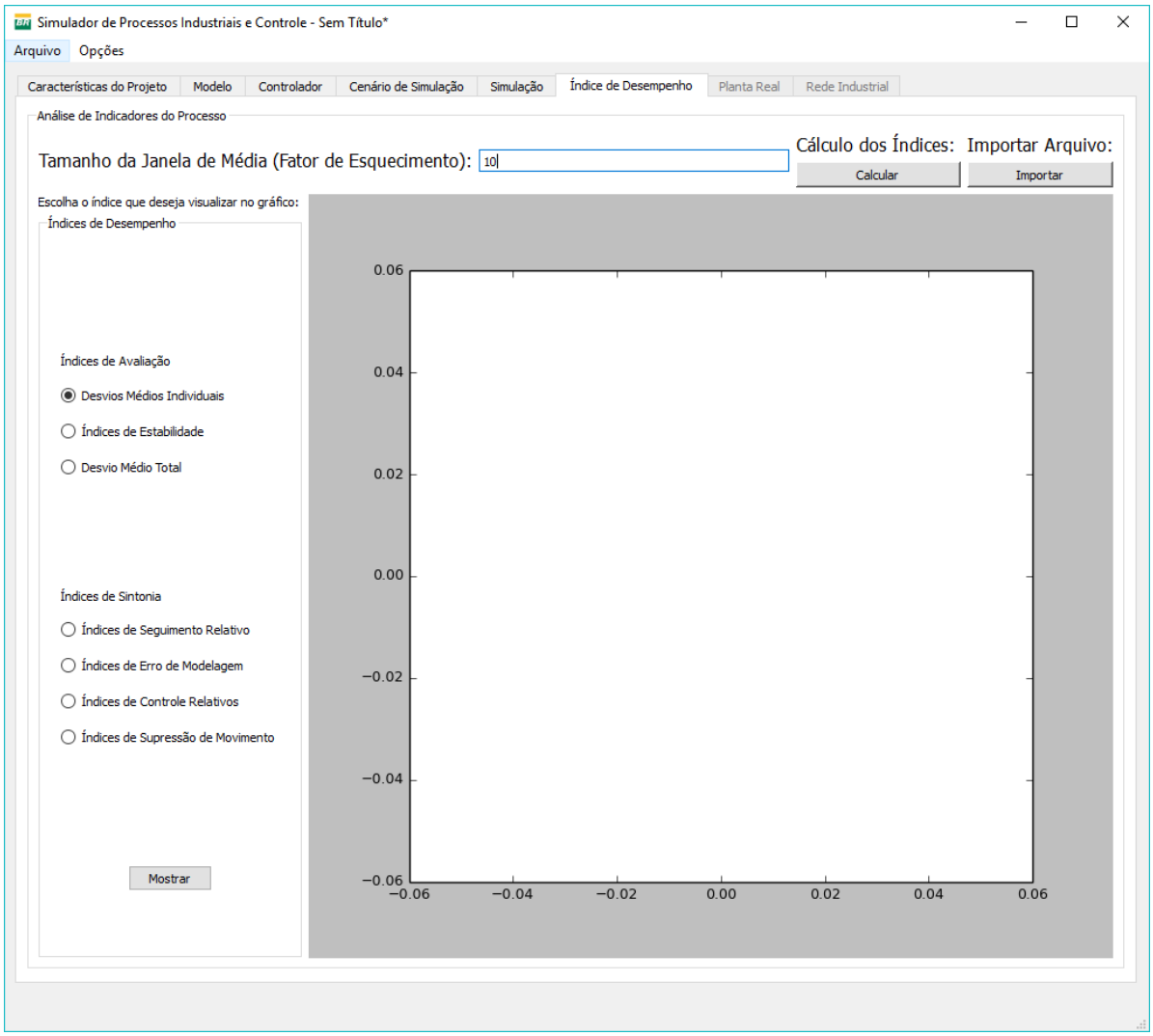

Figura 47 – Aba de índices de desempenho.

Existem duas formas de calcular os índices de desempenho. A primeira permite que uma simulação interna da interface tenha seus índices calculados e mostrados na interface, enquanto a outra possibilidade é importar para a interface um cenário de simulação externo.

Os índices calculados por simulações internas são feitos clicando no botão *Calcular*, o que abre duas janelas em sequência que perguntam ao usuário qual o cenário e qual o controlador que serão utilizados para calcular os índices. Após escolher o cenário e o controlador, basta ao usuário selecionar um dos índices situados na lateral esquerda da aba, clicando no botão "Mostrar". O índice de controle relativo não

é disponível para simulações internas porque alguns parâmetros necessários para o seu cálculo não foram implementados na interface.

Os arquivos importados apresentam um formato específico e têm uma extensão específica, chamada *.csv*. O usuário escolhe uma pasta contendo n arquivos *.csv*, cada um representando um instante de tempo da simulação externa e depois seleciona também um arquivo que contenha os ganhos de malha aberta do modelo do processo, geralmente na extensão *.mat*. Assim, os índices são calculados e segue-se o mesmo procedimento anterior para mostrar o resultado na interface. Na Figura 48 é mostrado o índice de seguimento relativo calculado a partir de uma simulação externa.

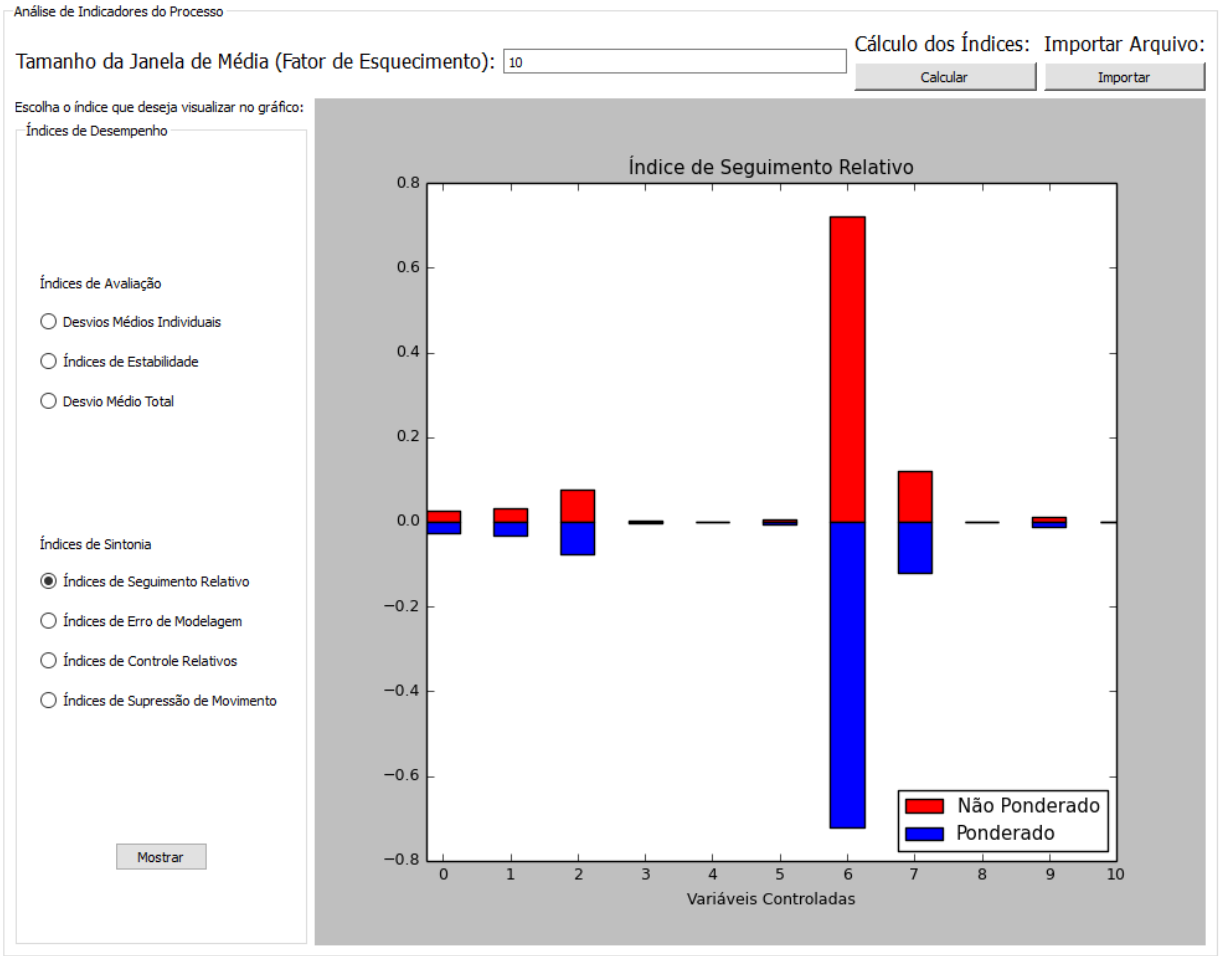

Figura 48 – Exemplo de arquivo importado para a interface.

5.7 Biblioteca de Modelos

Existem muitos processos plenamente estudados nas universidades. Sabendo disso foi concebida uma ferramenta com o intuito de possibilitar ao usuário simular esses processos que apresentam comportamento conhecido, oferecendo uma biblioteca com diferentes modelos armazenados, possibilitando a importação dos seus dados para a

interface. O foco da biblioteca são processos da indústria do petróleo e gás, visando especializar a ferramenta para processos petroquímicos comuns.

Para o usuário acessar essa biblioteca de modelos, foi criada na barra de ferramentas uma seção *Opções* que apertando em *Biblioteca* (ver Figura 25) uma nova janela é aberta, como pode ser visto na Figura 49. Nela estão presentes informações básicas sobre o modelo, além de uma breve descrição do processo, para apresentar ao usuário brevemente como o sistema se comporta caso este seja desconhecido.

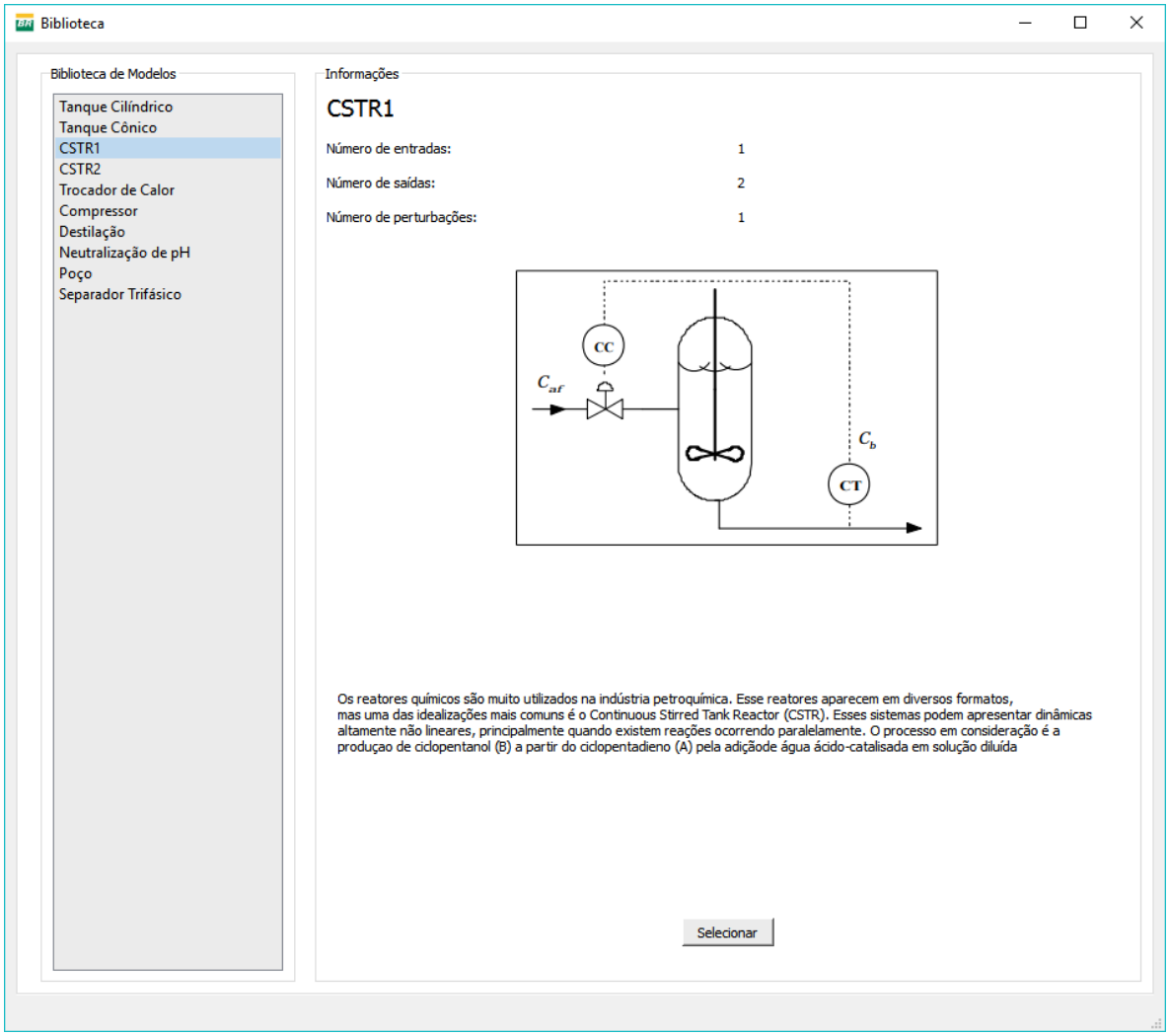

Figura 49 – Interface da biblioteca de modelos.

Dos modelos mostrados na Figura 49 foram implementados apenas o *Tanque Cônico*, *Tanque Cilíndrico* e *CSTR1*, pois os outros modelos requerem uma breve modificação na interface para permitir que sejam adicionadas equações estáticas para o cálculo de parâmetros do processo. Caso o usuário queira importar um modelo, ele deve primeiramente clicar no modelo na lista posicionada na parte esquerda da tela e então apertar no botão *Selecionar*.

Considerações Finais

Esse capítulo explicou detalhadamente cada elemento da interface e como eles funcionam. A interface foi criada para seguir uma sequência lógica e cronológica, sempre priorizando a usabilidade do usuário. Cada aba foi implementada individualmente, porém todas se comunicam constantemente, trocando informações relevantes para o bom funcionamento do sistema.

Algumas funções não foram detalhadas profundamente, como o controlador PID, pois foram focados os elementos principais da interface e que têm papel fundamental no seu desenvolvimento. As duas últimas abas foram desenvolvidas para outro projeto envolvendo controle preditivo aplicado a sistemas embarcados e por isso não são objetos de estudo dessa monografia.

6 Conclusão

A implementação do projeto conseguiu cumprir as expectativas impostas ao seu resultado final. Um dos objetivos era o desenvolvimento de uma interface gráfica capaz de simular processos industriais integrando com a sintonia de controladores preditivos para análises de malha fechada do processo, buscando uma ferramenta robusta mas que oferecesse ao usuário uma boa usabilidade. Outro objetivo cumprido foi a criação de um software com baixo custo de aquisição, pois a programação foi toda feita em Python, uma linguagem de programação extremamente eficiente, que diminui o esforço do programador e que possibilitou a implementação de uma ferramenta *open source*.

O produto da implementação foi uma interface gráfica acessível, mas conceitualmente complexa pelos elementos nela presentes, modelada seguindo os princípios da engenharia de software e com grande potencial. Parte desse potencial está nas diferentes áreas de atuação que a interface pode ser utilizada. Por exemplo, ela pode ser aproveitada na academia como uma ferramenta didática, os professores podem utilizá-la em disciplinas que estudam modelagem de processos industriais, controle preditivo, simulação em malha aberta e sistemas realimentados. Com isso, os alunos podem ter mais facilidade para adquirir conhecimentos nessas disciplinas, tendo uma maior familiaridade com os assuntos de forma mais intuitiva.

A interface também tem um potencial na indústria de processos, onde pode ser aproveitada para testes e simulações de sistemas complexos. Como dito anteriormente, é comum que a produção industrial seja contínua, necessitando o funcionamento constante da planta e interromper esse funcionamento pode gerar perdas financeiras consideráveis. Por isso, ferramentas que permitem a simulação de processos são muito importantes pois viabilizam que testes sejam realizadas sem a descontinuação da produção, ocorrendo em paralelo a ela. O que diferencia a ferramenta produzida é a possibilidade de uma simulação que envolva controladores avançados e de criar diversos cenários possíveis de simulação, analisando-os simultaneamente para verificar, por exemplo, a robustez do controlador em diferente situações.

Foi mostrado que é possível criar projetos diversos, possibilitando ao usuário formar um acervo de simulações armazenadas num disco rígido. Dessa forma, o engenheiro pode guardar num banco de dados informações referentes a momentos específicos da produção, gerando um banco de dados com o comportamento do sistema ao longo do tempo. Isso é útil para entender como o processo se comportou com o passar do tempo, permitindo a verificação de problemas quando algum processo foi alterado e até mesmo como reagir nessas situações analisando simulações passadas.

6.1 Trabalhos Futuros

Mesmo com a finalização da interface gráfica, algumas funcionalidades ainda poderiam ser adicionadas para melhorar a performance do sistema. Assim, são sugeridos alguns temas que poderiam ser objeto de estudo para implementação na interface:

- Referentes ao MPC:
  a) Permitir a definição de diferentes funções objetivo para melhorar o ajuste do MPC;
  b) Adicionar também técnicas de controle adaptativos;
  c) Adicionar mais algoritmos preditivos, como o DTC-GPC;
- Interface do usuário:
  a) Adição de técnicas de identificação de sistemas para auxiliar o usuário na obtenção do modelo do processo;
  b) Adição de mais modelos na biblioteca;
  c) Adição de diagramas de bloco para melhorar criação de cenários;
  d) Cálculo de equações estáticas.

Os pontos propostos não alterariam consideravelmente a interface atual, apenas adicionaria novas funcionalidades para incrementar ao sistema, sempre pensando na usabilidade da interface e mantendo a qualidade do produto.

Referências

1 BEAUBIEN, J. M.; BAKER, D. P. The use of simulation for training teamwork skills in health care: how low can you go? *Qual Saf Health Care*, v. 13, p. i51–i56, 2004. Citado na página 17.

2 GARCIA, C. *Modelagem e Simulação de Processos Industriais e de Sistemas Eletromecânicos*. 2. ed. [S.l.]: Universidade de São Paulo, 2005. Citado 2 vezes nas páginas 17 e 18.

3 LIMA, D. M. *Sistema Embarcado de Controle Preditivo para Processos Industriais*. [S.l.], 2013. Citado 8 vezes nas páginas 17, 32, 54, 56, 57, 58, 59 e 76.

4 LABAKI, J.; CORREA, P. *Introdução ao Python*. Disponível em: <http://www.sr.ifes.edu.br/~secchin/Programacao/Python/python.html>. Citado na página 18.

5 VENNERS, B. *The Making of Python: A Conversation with Guido Van Rossum, Part I*. Disponível em: <http://www.artima.com/intv/pythonP.html>. Citado na página 18.

6 LIMA, D. M. *Predictor Based Robust Control of Dead Time Processes*. [S.l.], 2014. Citado 7 vezes nas páginas 21, 23, 24, 26, 30, 35 e 38.

7 CAMACHO, E. F.; BORDONS, C. *Model Predictive Control*. 2. ed. [S.l.]: Springer, 2004. Citado na página 22.

8 LELIć, M. A.; ZARROP, M. B. Generalized pole-placement self-tuning controller part 1, basic algorithm. *International Journal of Control*, v. 46, p. 369–374, 1987. Citado na página 22.

9 LINKERS, D.; MAHFONF, M. *Advances in model-based predictive control*. [S.l.]: Oxford University Press, 1994. Citado na página 22.

10 CLARKE, D. W. Application of generalized predictive control to industrial processes. *IEEE Control Systems Magazine*, v. 8, p. 49 – 55, 1988. Citado na página 22.

11 RICHALET, J. Industrial applications of model based predictive control. *Automatica*, v. 29, p. 1251–1274, 1993. Citado na página 22.

12 RICHALET, J.; AL et. Model predictive heuristic control: Applications to industrial processes. *Automatica*, v. 14, p. 413–428, 1978. Citado 2 vezes nas páginas 22 e 30.

13 CAMACHO, E. F.; NORMEY-RICO, J. E. *Control of Dead-Time Processes*. 1. ed. [S.l.]: Springer, 2007. Citado 2 vezes nas páginas 22 e 31.

14 TAKATSU, H.; ITOH, T.; ARAKI, M. Future needs for the control theory in industries. report and topics of the control technology survey in japanese industry. *Journal of Process Control*, v. 8, p. 369–374, 1998. Citado na página 23.

15 CUTLER, C.; RAMAKER, B. Dynamic matrix control–a computer control algorithm. *Joint Automatic Control Conference*, v. 17, p. 72, 1979. Citado na página 30.

16 LEE, J. H.; MORARI, M. State-space interpretation of model predictive control. *Automatica*, v. 30, p. 707–717, 1994. Citado na página 30.

17 DATTA, A.; OCHOA, J. Adaptive internal model control: Design and stability analysis. *Automatica*, v. 32, p. 261–266, 1996. Citado na página 30.

18 CLARKE, D.; MOHTADI, C.; TUFFS, P. Generalized predictive control - part i. the basic algorithm. *Automatica*, p. 167–148, 1987. Citado na página 30.

19 KAYSER, R. M. C. D.; CUAWENBERGHE, A. van. Extended prediction self-adaptive control. *IFAC Symposium on Identification and System Parameter Estimation*, p. 1317–1328, 1985. Citado na página 30.

20 PLUCENIO, A. A practical approach to predictive control for nonlinear processes. *IFAC Symposium on Identification and System Parameter Estimation*, 2007. Citado 2 vezes nas páginas 30 e 38.

21 SKOGESTAD, S.; POSTLETHWAITE, I. *Multivariable Feedback Control: Analysis and Design*. 2. ed. [S.l.]: John Wiley, 2005. Citado na página 31.

22 LIMA, D. M. et al. Improving robustness and disturbance rejection performance with industrial mpc. *Anais do XX Congresso Brasileiro de Automática*, 2014. Citado na página 37.

23 GROUP, O. M. *Unified Modelling Language(UML) Resource Page*. Disponível em: <http://www.uml.org/>. Citado na página 45.

24 LEITE, A. *Metodologia de Desenvolvimento de Software*. Disponível em: <http://www.devmedia.com.br/metodologia-de-desenvolvimento-de-software/1903>. Citado na página 45.

25 BARBOSA, W. L. de O.; PASCO, C. D. *Processo Unificado e Processo Unificado Racional*. 2011. Disponível em: <http://www.webartigos.com/artigos/processo-unificado-e-processo-unificado-racional-up-e-rup/65404/>. Citado na página 46.

26 PRESSMAN, R. *Engenharia de Software - 7.ed.:*. [S.l.]: McGraw Hill Brasil, 2009. Citado na página 46.

27 WAZLAWICK, R. S. *Análise e projeto de sistemas de informação orientados a objetos*. 2. ed. [S.l.]: Campus, 2011. Citado 7 vezes nas páginas 47, 48, 49, 50, 52, 57 e 70.

28 LARMAN, C. *Applying UML and Patterns. An Introduction to Object-Oriented and Design and The Unified Process*. 3. ed. [S.l.]: Prentice Hall, 2004. Citado 3 vezes nas páginas 47, 48 e 49.

29 LEAL, F.; ALMEIDA, D. A. de; MONTEVECHI, J. A. B. Uma proposta de tÉcnica de modelagem conceitual para a simulaÇÃo atravÉs de elementos do idef. *XL SBPO – A Pesquisa Operacional e o uso racional de recursos hídricos*, 2008. Citado na página 51.

30 RICARTE, I. L. M. *Herança*. 2000. Disponível em: <http://www.dca.fee.unicamp.br/courses/PooJava/heranca/index.html>. Citado na página 57.

31 ROCHA, G.; FILHO, H.; JURITY, R. Camada de persistência de dados para aplicações java: O hibernate. *Instituto de Matemática – Universidade Federal da Bahia*, 2009. Citado na página 59.

32 W3C. *Extensible Markup Language (XML)*. 2008. Disponível em: <https://www.w3.org/TR/REC-xml/>. Citado na página 59.

33 JONES, C. A.; DRAKE, J. F. L. *Python and XML*. 1. ed. [S.l.]: O'Reilly, 2002. Citado 3 vezes nas páginas 59, 61 e 62.

34 MAGNUM. *A História do Python*. 2014. Disponível em: <http://mindbending.org/pt/a-historia-do-python>. Citado na página 62.

35 DOXYGEN. *Doxygen*. 2015. Disponível em: <http://www.stack.nl/~dimitri/doxygen/>. Citado na página 62.

36 PYXB. *PyXB: Python XML Schema Bindings*. 2009. Disponível em: <http://pyxb.sourceforge.net/>. Citado na página 79.

37 CORTEZ, P. E. F. et al. Performance indexes for assistance in retuning multivariable model predictive controllers. *Industry Applications (INDUSCON), 2014 11th IEEE/IAS International Conference*, 2014. Citado 2 vezes nas páginas 106 e 107.

Apêndices

APÊNDICE A – Documentos do Projeto de *Software*

A.1 Requisitos

Tabela 5 – Descrição do requisito Definição dos Parâmetros do Projeto

F1 – Definição dos Parâmetros do Projeto
Descrição – a interface gráfica deve permitir que o usuário defina os parâmetros do projeto, como número de saídas, número de entradas, número de perturbações, nome do projeto, tempo de amostragem.
Restrições Não-Funcionais
NF1.1 – usuário só pode definir valores inteiros para o número de entradas, saídas e perturbações.
NF1.2 – usuário só pode definir valores inteiros positivos para o tempo de amostragem.
NF1.3 – o nome do projeto tem um tamanho máximo de 12 caracteres.
NF1.4 – só pode ser iniciado um projeto

Tabela 6 – Descrição do requisito Definição dos Modelos

F2 – Definição dos Modelos
Descrição – a interface gráfica deve permitir ao usuário criar, remover ou modificar modelos.
Restrições Não-Funcionais
NF1.1 – usuário não pode criar modelos com nome já existente na lista de modelos.
NF1.2 – definição de diferentes tipos de modelos

Tabela 7 – Descrição do requisito Definição dos Controladores

F3 – Definição dos Controladores
Descrição – a interface gráfica deve permitir ao usuário criar, remover ou modificar controladores.
Restrições Não-Funcionais
NF1.1 – usuário não pode criar controladores com nome já existente na lista de controladores.
NF1.2 – definição de diferentes tipos de controlador

Tabela 8 – Descrição do requisito Configuração do Cenário de Simulação

F4 – Configuração do Cenário de Simulação
Descrição – a interface gráfica deve permitir ao usuário criar cenários de simulações, configurando-os conforme sua necessidade. A configuração engloba definição de referências, perturbações (entradas em malha aberta), número de iterações da simulação, os modelos e controladores a serem simulados e condições iniciais das variáveis.
Restrições Não-Funcionais
NF1.1 – usuário não pode criar cenários com nome já existente na lista de cenários.
NF1.2 – número de iterações deve ser um número inteiro positivo
NF1.3 – condições iniciais devem ser números reais
NF1.4 – referências e perturbações podem ser do tipo degrau, rampa, senoidais ou ruídos branco
NF1.4 – cenário pode ser simulado em malha aberta ou malha fechada

Tabela 9 – Descrição do requisito Visualização dos Gráficos de Simulação

F5 – Visualização dos Gráficos de Simulação
Descrição – a interface gráfica deve permitir ao usuário verificar os resultados da simulação na forma de gráficos, de maneira clara e simples.

Tabela 10 – Descrição do requisito Calcular Ação de Controle

F6 – Calcular Ação de Controle
Descrição – o algoritmo de simulação deve, de forma eficiente quando o cenário estiver configurado corretamente, calcular as ações de controle, otimizando uma função custo do tipo quadrática para o cálculo utilizando o algoritmo MPC.
Restrições Não-Funcionais
NF1.1 – caso ocorra de o tempo de cálculo ultrapassar o tempo de amostragem seguidamente, um alarme deve ser acionado.
NF1.2 – caso ocorram problemas de otimização, deve-se tomar providências para não causar problemas de estabilidade.
NF1.3 – opção de vários algoritmos MPC.

Tabela 11 – Descrição do requisito Calcular Índices de Desempenho

F7 – Calcular Índices de Desempenho
Descrição – o algoritmo deve ser capaz de calcular os índices de desempenho de uma dada simulação, analisando a robustez do sistema.
Restrições Não-Funcionais
NF1.1 – usuário pode calcular os índices através de uma simulação realizada na própria interface ou pode exportar dados a partir de arquivo .xsd com um formato específico que contem dados de uma simulação externa.
NF1.2 – Alguns índices são exclusivos de simulações exportadas, pois alguns índices apresentam variáveis que não serão implementadas na interface.

Tabela 12 – Descrição do requisito Implementação da Simulação

F8 – Implementação da Simulação
Descrição – o algoritmo deve fazer os cálculos de saída a partir de ações de controle calculadas anteriormente, de forma a mostrar o resultado da simulação de forma correta e coerente.

Tabela 13 – Descrição do requisito Gerenciamento do Banco de Dados

F9 – Gerenciamento do Banco de Dados
Descrição – o algoritmo deve gerenciar o banco de dados de forma a ser capaz de salvar ou abrir documentos XML para o uso, armazenando as informações necessárias para que o usuário possa reutilizar um projeto criado posteriormente.
Restrições Não-Funcionais
NF1.1 – os dados guardados devem seguir o padrão definido no schema
NF1.2 – caso alguma informação esteja incorreta, gerar uma janela de aviso

Tabela 14 – Descrição do requisito Gerenciar Dados do Processo

F10 – Gerar Dados do Processo
Descrição – algoritmo necessário para gerenciar todos os dados internos, como listas das variáveis, dos modelos, controladores, cenário, além de dados essenciais como tempo de amostragem, referências, perturbações, etc.

Tabela 15 – Descrição do requisito Atualizar Dados do Processo

F11 – Atualizar Dados do Processo
Descrição – o sistema deve ser capaz de atualizar os dados do processo sempre que algo for alterado no projeto, de forma a manter os dados armazenados corretamente e com coerência às ações do usuário.
Restrições Não-Funcionais
NF1.1 – dados armazenados de forma inválido não devem ser atualizados.

Tabela 16 – Descrição do requisito Salvar Dados da Simulação

F13 – Salvar Dados da Simulação
Descrição – o sistema deve ser capaz de armazenar os resultados das simulações para que seja possível a plotagem dos gráficos de simulação.
Restrições Não-Funcionais
NF1.1 – os dados devem ser salvos em listas, com tamanho igual ao número de saídas do processo.

Tabela 17 – Descrição do requisito Abrir Dados de Simulações Armazenadas

F14 – Abrir Dados de Simulações Armazenadas
Descrição – o sistema deve ser capaz de ler os resultados das simulações para que seja possível a plotagem dos gráficos de simulação.
Restrições Não-Funcionais
NF1.1 – caso algum cenário não tenha sido simulado, gerar uma janela de aviso caso o sistema tente abrir os dados de simulação deste cenário.