UNIVERSIDADE DE SÃO PAULO
ESCOLA POLITÉCNICA
DEPARTAMENTO DE ENGENHARIA MECATRÔNICA

FELIPE VALLIM ALMIRALL

LEONARDO PAPACIDERO GODOY

INDÚSTRIA 4.0: PROJETO E INTEGRAÇÃO DE GÊMEO DIGITAL DE BANCADA DIDÁTICA E EMULADOR DE REDE DE PETRI

TRABALHO DE CONCLUSÃO DE CURSO

SÃO PAULO
2021

FELIPE VALLIM ALMIRALL

LEONARDO PAPACIDERO GODOY

INDÚSTRIA 4.0: PROJETO E INTEGRAÇÃO DE GÊMEO DIGITAL DE BANCADA DIDÁTICA E EMULADOR DE REDE DE PETRI

Trabalho de Conclusão de Curso apresentado ao Departamento de Engenharia Mecatrônica da Universidade de São Paulo, como requisito parcial para a obtenção do título de Engenheiro.

Orientador: Prof. Dr. Fabrício Junqueira
Escola Politécnica da USP

SÃO PAULO
2021

Este trabalho é dedicado às nossas familias, aos nossos professores e aos nossos amigos.

AGRADECIMENTOS

Gostaríamos de dedicar um agradecimento especial ao Professor Dr. Fabrício Junqueira, por todo o apoio e paciência ao longo de toda a graduação mas, principalmente, ao longo do desenvolvimento deste trabalho.

Agradecemos à FESTO, por disponibilizarem o modelo 3D da bancada para uso no Gêmeo Digital deste projeto.

Agradecemos também a todos aqueles que um dia nos ensinaram. De forma intencional ou não nos fizeram crescer, nos fizeram capazes de abraçar dúvidas cada vez maiores. A vocês, nossa gratidão, por terem nos dado a faísca da curiosidade que hoje guia nossos raciocínios e a nossa vontade de sermos melhores a cada dia.

Aos familiares e amigos que compartilharam de nossas conquistas, mas que também nos seguraram em nossos tropeços. Sem vocês, entrar nessa escola jamais teria sido uma opção. Com vocês, sair dela com o diploma é nossa forma de dizer: Conseguimos.

If you can't explain it simply, then you don't understand it well enough. (Autor Desconhecido).

RESUMO

ALMIRALL, Felipe e GODOY, Leonardo. Indústria 4.0: Projeto e Integração de Gêmeo Digital de bancada didática e emulador de rede de Petri. 2021. 75 f. Trabalho de Conclusão de Curso – Departamento de Engenharia Mecatrônica, Universidade de São Paulo. São Paulo, 2021.

Palavras-chave: Emulador de rede de Petri, Gêmeo digital, Simulação de produção, Indústria 4.0

O presente trabalho desenvolve um programa capaz de interpretar arquivos de redes de Petri desenvolvidos em PIPE 4.3, utilizando essa estrutura para comandar sinais digitais de um Gêmeo Digital de uma bancada didática, via protocolo OPC-UA. Demonstra a estrutura construída para o funcionamento do interpretador e controlador através de diagramas UML utilizando linguagem de programação Orientada a Objeto, bem como a elaboração do Gêmeo Digital da bancada didática de forma fidedigna através do uso do Software Visual Components. Estabelece a conexão do Servidor desenvolvido com o Cliente no Gêmeo Digital via OPC-UA com sucesso e propõe diferentes usos da ferramenta em sala de aula.

ABSTRACT

ALMIRALL, Felipe e GODOY, Leonardo. Digital Twin: PLC Simulation. 2021. 75 f. Trabalho de Conclusão de Curso – Departamento de Engenharia Mecatrônica, Universidade de São Paulo. São Paulo, 2021.

Keywords: Petri net Emulator, Digital Twin, Production Simulation, Industry 4.0.

This project develops a program capable of interpreting Petri net files developed in PIPE 4.3 and uses this structure to command digital signals from a digital twin of a didactic station, via OPC-UA protocol. It demonstrates the structure built for the operation of the interpreter and controller through UML diagrams using Object Oriented programming language, as well as the elaboration of the Digital Twin of the didactic station in a reliable way through the use of Visual Components Software. It successfully connects the Server developed with the Client in the Digital Twin via OPC-UA and proposes different uses of the tool in classes.

SUMÁRIO

1 Introdução

O termo "Indústria 4.0" encontra-se altamente difundido no cenário hodierno. Ele foi primordialmente definido pelo Governo Federal Alemão como uma estrutura emergente, na qual a manufatura e logística são estruturadas na forma de sistemas de produção físico-cibernéticos, que usam intensivamente as informações disponíveis globalmente e as redes de comunicações para uma troca amplamente automatizada de dados, na qual processos de produção e negócios são combinados (BAHRIN et al., 2016). Isso remete à mudança de paradigma atual na estrutura, tecnologia e organização da indústria moderna, denominada como a 4ª Revolução Industrial. De acordo com (LASI et al., 2014) este processo é caracterizado por uma avançada digitalização na indústria, a combinação de tecnologias de Internet e tecnologias voltadas na aplicação de objetos "inteligentes" (máquinas e produtos).

A Indústria 4.0 possui 9 pilares principais que permitem a transformação de módulos de produção isolados em uma produção totalmente integrada, automatizada e otimizada (VAIDYA; AMBAD; BHOSLE, 2018). Estes são "Big Data and Analytics", "Autonomous Robots", "Simulation", "System Integration", "The Industrial Internet of Things", "Cyber security and Cyber Physical Systems", "The Cloud", "Additive Manufacturing" e "Augmented Reality" (VAIDYA; AMBAD; BHOSLE, 2018). Com base nos campos apresentados o projeto que será desenvolvido possui um enfoque no pilar de "Simulação".

A simulação compreende uma representação aproximada da realidade, operação de um processo ou sistema ao longo de um certo tempo. É uma ferramenta poderosa para a avaliação e análise de projetos de novos sistemas, modificações em sistemas existentes e alterações propostas para sistemas de controle e regras operacionais (II, 2004). Esse pilar está ligado diretamente ao termo Gêmeo Digital, conceito essencial da Indústria 4.0 e que embasa e fundamenta o projeto que será discorrido posteriormente.

Gêmeo Digital depreende o conceito de simulação em uma forma mais específica, é uma tecnologia que virtualiza um ambiente ou sistema físico para que a organização simule cenários e gere valor agregado aos negócios. De acordo com (QUINALHA, 2018) "Gêmeo digital pode ser definido como um modelo digital de um objeto real, que representa sua configuração física com riqueza de detalhes suficiente ou até mesmo com simplificações pertinentes, alimentado por dados de sensores, o que ilustra a situação instantânea deste objeto no mundo real. Um gêmeo digital pode representar um ativo individual, um sistema composto por ativos diferentes

ou um conjunto de vários ativos idênticos."

Tendo em vista esse conceito, o Gêmeo Digital mostra-se uma poderosa e relevante ferramenta para a didática. Isso se mostra verídico e pertinente em (NIKOLAEV et al., 2018) que valida o benefício da utilização do Gêmeo Digital na educação de alunos de mestrado em ciência aplicada através da metodologia de desenvolvimento de produto baseada em simulação. O procedimento educacional foi construído em torno de um processo real de desenvolvimento de produto, um pequeno "Veículo Aéreo Não Tripulado". Foram utilizadas diversas ferramentas de simulação no processo educacional com o intuito de criar o Gêmeo Digital do produto alcançando ótimos resultados no ensino.

Ademais, esse tema é citado em (SEPASGOZAR, 2020) no qual é demonstrada a aplicação de tecnologias de Gêmeo Digital e da Realidade Virtual e Aumentada para o ensino de estudantes de arquitetura, engenharia e construção. Este artigo desenvolveu um conjunto de módulos virtuais e discutiu sua aplicabilidade na área da educação na construção civil, por meio de um curso prático como um estudo de caso. Um dos módulos implementou um Gêmeo Digital de escavadeira que estava vinculado a uma escavadeira real, com isso os alunos poderiam usá-la para aprender diferentes movimentos da escavadeira. A implementação de tecnologia virtual exclusiva e o *feedback* dos alunos mostram que o uso de ferramentas virtuais *online* para aprender cursos práticos de construção é viável e útil.

Levando em consideração as ideias e referências citadas, o Gêmeo Digital torna-se um potente instrumento das instituições de ensino na área da engenharia para elucidar de forma prática, clara e visual conceitos e processos de novos conhecimentos. Já no campo da indústria e negócios permite o treinamento e desenvolvimento de colaboradores sobre os processos e tecnologias empregadas na produção. Há a possibilidade de estudo, análise, viabilidade e otimização dos recursos e procedimentos aplicados ou até que serão aplicados, ou seja, permite-se avaliar antes de adquirir ou instalar um processo ou tecnologia, se este será pertinente e atenderá os requisitos estimados pela entidade.

Com uma simulação funcional (que representa de forma satisfatória e adequada as características do sistema real simulado), flexível e com aspectos gráficos, permite-se um entendimento mais claro e objetivo dos processos e composições assimiladas no simulacro. Há uma grande facilidade de acesso e estudo pois não é necessário possuir os equipamentos, instalações físicas, ou estar presente em algum local o que é de enorme vantagem para a escalabilidade de uma aula ou de um treinamento, principalmente no contexto atual de pandemia do Covid-19.

1.1 Objetivo

Levando em consideração as motivações didáticas e tecnológicas mencionadas em 1 e a partir de um contexto de Indústria 4.0 é posto o objetivo deste trabalho. O projeto consiste no projeto e implementação de um Gêmeo Digital de uma bancada MPS da Festo e de um emulador de rede de Petri e integração destes dois. O emulador receberá como entrada um modelo gráfico em forma de rede de Petri que representa e descreve o controle da bancada. Será estabelecido uma comunicação entre o emulador e o Gêmeo Digital da bancada MPS permitindo a troca de informações entre estes e garantindo o controle do módulo simulado.

Sendo assim, com a obtenção do Gêmeo Digital e do emulador mencionados, adquire-se uma potencial ferramenta para ser utilizada posteriormente para didática de forma prática e funcional e para uma melhor compreensão de conceitos e processos da Indústria 4.0.

1.2 Estrutura do texto

O texto vigente é composto por 6 capítulos. Sendo o capítulo atual responsável pela introdução que contextualiza o tema do estudo, expõe as motivações e o objetivo do trabalho.

O capítulo 2 contém a Revisão Bibliográfica do estudo. Exibe-se diferentes trabalhos e estudos publicados acerca do tema tratado, contextualizando e apresentando a conjuntura de conteúdos em vigência relacionados a este trabalho. Ademais é exposto algumas possíveis tecnologias para a aplicação e as escolhidas para a execução do objetivo proposto. Também inclui-se uma fundamentação teórica acerca da rede de Petri e a apresentação da bancada MPS Festo que será tratada.

O capítulo 3 aborda o projeto do Gêmeo Digital da bancada MPS bem como a metodologia utilizada para tal e também o projeto do emulador de Rede de Petri juntamente com sua metodologia baseada no UML (*Unified Modeling Language*) para a elaboração da estrutura do programa interpretador de arquivo XML (*Extensible Markup Language*).

Já no capítulo 4 exibe-se os processos e passos da implementação do Gêmeo Digital da Bancada e o emulador de rede de Petri. Além do mais explícita e descreve a integração destas duas partes.

Por sua vez, o capítulo 5 documenta os resultados das implementações e também a avaliação dos testes feitos.

Por fim, o capítulo 6 comporta as conclusões do trabalho exposto e possíveis aplicações futuras dos recursos construídos.

2 Revisão Bibliográfica

Este capítulo inicia-se com Estado da Arte a respeito do tema do trabalho. É exibido diversos trabalhos e estudos acerca de Gêmeo Digital, seu contexto atual, pertinência, cenários e aplicações, bem como seu uso e relevância como ferramenta didática. Há também estudos que se relacionam com a temática de emuladores de rede de Petri e observações baseadas nos estudos levantados.

Em seguida são levantadas possíveis tecnologias, como *softwares*, linguagens de programação e dispositivos de processamento para a implementação do projeto. Também, dentre as opções pontuadas, é apresentado os recursos escolhidos e os motivos de tal.

Prosseguindo, é posto uma fundamentação teórica acerca da linguagem de modelagem baseada em elementos gráficos chamada de rede de Petri, a qual é um componente fundamental para a aplicação deste trabalho, já que esta descreve o controle aplicado no projeto do Gêmeo Digital criado.

Ademais, expõe-se a respeito das bancadas MPS Festo assim como a escolha de uma delas para dar prosseguimento ao projeto do Gêmeo Digital. Discorre-se desta escolhida de forma detalhada, descrevendo seu funcionamento e componentes.

Por fim, é feito uma síntese do capítulo de Revisão Bibliográfica apontando alguns comentários e fazendo a correlação com o próximo capítulo.

2.1 Estado da Arte

O Conceito de Gêmeo Digital surge como uma simulação dinâmica em tempo real de um sistema físico e seus recursos envolvidos (BERISHA; CARUSO; HARTEIS, 2021), o que, combinado a representações detalhadas do ambiente em que o sistema está inserido, viabiliza uma representação digital mais precisa do sistema que reflete, permitindo análises e controle (PANETTA, 2016).

Empresas estão pesquisando e implementando novas tecnologias para melhorar o treinamento de funcionários, a eficiência do processo de produção, a redução de custos e a qualidade do produto (OSBORNE; MAVERS, 2019). O impacto de melhorias, mesmo que pequenas, numa linha de produção reflete no agregado a um ganho de produtividade que pode impactar a quantidade de produto finalizada no dia de uma empresa. Simulações, entram nesse

contexto, para que uma empresa possa alterar os parâmetros atuais de sua cadeia de produção e testar novas oportunidades sem que a cadeia de produção seja interrompida. Para tais simulações, as melhorias no processo serão tão fiéis quanto o modelo representar a realidade.

Além da produtividade, pode-se considerar que uma emulação virtual de um sistema em tempo real abre portas para: (i) a manutenção a distância do sistema, (ii) a monitoração do processo, (iii) a aquisição de dados (GöKALP et al., 2016) (iv) para a definição de pontos de ociosidade, de gargalo ou até manutenção preventiva, (v) o cruzamento de previsões de demanda com a possibilidade de entrega e até (vi) a previsão e prevenção de cenários críticos que possam acarretar em desligamento do sistema como um todo. Sabendo disso, uma simulação pode alterar a velocidade com que um processo ocorre, bem como aumentar o nível de detalhamento visível para que a pessoa que a acompanha possa interagir sabendo o que deve ou não ser reproduzido em um ambiente real, principalmente por conta de uma resposta em tempo real de determinado *input* sem que haja danificação da cadeia produtiva.

Em um ambiente simulado, não só o acompanhamento do maquinário é possível, mas também a simulação de cenários distintos ao real, o que é extremamente conveniente para a apresentação do sistema a pessoas não familiarizadas. Essa aplicação é válida tanto para o treinamento de funcionários quanto para aplicação didática em sala de aula e pode ocorrer por meio de diferentes tecnologias, mas sempre com foco na representação virtual da realidade, possibilitando o engajamento de aprendizados de alto nível (SILVIA, 2012).

A pertinência da aplicação de Gêmeo Digital no âmbito educacional pode ser vista em (SEPASGOZAR, 2020). O estudo contempla a utilização de tecnologias digitais para a didática de alunos da área AEC (arquitetura, engenharia e construção civil). A inovação deste projeto advém do desenvolvimento de módulos de construção imersiva, práticas de implementação do ensino digital e apresentação da capacidade das tecnologias virtuais para a educação. Para tanto, foi utilizado um curso de construção como caso de uso e foram implementados módulos criados para a formação do aluno. Um dos módulos aplicados foi baseado em um Gêmeo Digital de escavadeira que estava vinculada a uma entidade física da máquina, possibilitando que os alunos pudessem manejá-la para aprender os diferentes movimentos da escavadeira.

Nessa oportunidade, verificou-se que em processos complicados na construção, como perfuração e sondagem subterrânea - nos quais os alunos não possuíam nenhum conhecimento prévio -, os novos métodos de ensino digital mostraram-se superiores aos métodos de aprendizagem tradicionais, como livros didáticos. A possibilidade de praticar em um ambiente simulado, que permite aos alunos correção e repetição para melhorar suas habilidades com falhas sem

risco, mostrou-se um ótimo mecanismo didático. Como resultado do estudo, a satisfação dos alunos aumentou, bem como a apreciação e o aprendizado, o que foi refletido no *feedback* dos alunos.

A aplicação de Gêmeo Digital também se mostra relevante para o treinamento e segurança em empresas. Tradicionalmente, o treinamento de segurança tem sido realizado por aulas teóricas em uma sala de aula externa ou treinamento no local na instalação de produção. Aulas como método de treinamento de segurança não oferecem a possibilidade de experiência prática, com estudos anteriores indicando a ineficácia do treinamento de segurança tradicional em comparação com o treinamento de segurança em um ambiente virtual. (SIM et al., 2019), (LE; PEDRO; PARK, 2015).

Não obstante, frequentemente é necessária uma pausa na produção para realizar o treinamento de segurança dentro das células de trabalho, o que pode causar estresse ao instrutor e instruídos, além de aumentar a exposição a riscos. Logo, do ponto de vista da empresa, o treinamento utilizando tecnologias como Gêmeo Digital, mostra-se vantajoso, em vista da não necessidade de pausar algum módulo da produção para sua aplicação. (SIM et al., 2019)

Em (KAARLELA; PIESKä; PITKäAHO, 2020) o assunto sobre aplicação da tecnologia do Gêmeo Digital no treinamento de segurança também é abordado. Discorre-se sobre sua implementação em um laboratório de robótica, possibilitando a instrução dos alunos e visitantes sobre os procedimentos de segurança antes de entrar fisicamente no laboratório. Durante o treinamento, os participantes adquirem conhecimento de como controlar robôs, suas trajetórias de movimento, áreas de alcance e seus dispositivos de segurança específicos. Ademais, o autor levanta a necessidade de métodos inovadores de treinamento em segurança em vários setores da indústria, especialmente o de construção e o de mineração. Além do mais, ressalta-se benefícios da aplicação desta tecnologia nesse contexto, como: (i) viabilidade de acesso remoto ao curso, (ii) possibilidade de participação de um público mais amplo de vários locais de trabalho no treinamento, (iii) possibilidade da análise de risco e planejamento de segurança nas fases iniciais de projeto de uma nova célula de produção e (iv) fornecimento de aprendizado quase prático de diferentes tarefas de trabalho em tempo real, sem entrar fisicamente nas instalações de produção. Portanto, o Gêmeo Digital é uma forma promissora de desenvolver o treinamento de segurança para ser mais ilustrativo e eficiente. O aprendizado virtual oferece um método intuitivo, seguro e sem estresse para tais treinamentos.

Levando em conta cenários de risco possíveis de serem simulados sem consequências reais, em (EL-GENK et al., 2019) vê-se uma implementação de uma emulação de CLP

(Controlador lógico programável) de um contexto bastante crítico. A realização do projeto tem como objetivo final a simulação de um CLP inserido em um contexto de usina nuclear e, ainda que as motivações estejam concentradas no campo da segurança digital da emulação versus a segurança de um CLP convencional, são evidentes as preocupações com a segurança de cenários críticos passíveis de simulação e testes em um escopo mais abrangente. O projeto de El-Genk, na ocasião, utiliza um Raspberry Pi com interface TCP/IP conectada a um PC Servidor para emular a conexão do CLP no Cliente.

Ainda no contexto de segurança digital, vê-se no estudo grande preocupação com a segurança da informação. Obviamente que um fator nevrálgico em se tratando de uma usina nuclear, mas que também deve ser foco em qualquer linha de produção conectada no contexto da Indústria 4.0. Com as máquinas conectadas à rede, para que haja acesso remoto e, eventualmente, processamento dos dados, surge a preocupação para que protocolos de segurança sejam suficientemente seguros e rígidos, a fim de não expor dados sensíveis e segredos comerciais da indústria. Assim, é pré-requisito que a troca de informação ocorra de maneira segura. Estabeleceu-se, em 1994, um protocolo padrão chamado de OPC-UA para esta aplicação, e por estes motivos, além de sua universalidade no contexto da Indústria 4.0, passa a ser um protocolo de grande interesse para o uso em projetos envolvendo o uso de Gêmeo Digital.

Abordando o problema que é emular um CLP, deve ser levado em consideração que "usar uma emulação para avaliar o comportamento de uma rede de CLPs é difícil devido à falta de ferramentas que imitem com precisão o comportamento em tempo real de tais redes" (BABU; NICOL, 2016). Isso pois, o projeto demanda baixo atraso em resposta, bem como a capacidade de processamento assíncrono, motivo pelo qual uma representação de rede de Petri ganha destaque e passa a ser considerada.

Ademais, verifica-se que a utilização de um emulador de rede de Petri como forma de estabelecer o controle de eventos discretos empreendidos pela bancada MPS Festo se mostra uma opção para validar o projeto de controle da bancada sem necessariamente envolver a programação por intermédio dos programas propostos pela fabricante. A rede de Petri depreende uma linguagem de modelagem gráfica de sistemas distribuídos discretos. De acordo com (BARROS, 2006) essa linguagem possui benefícios e vantagens pois consiste em uma representação gráfica simples (círculos, retângulos e setas), representação algébrica simples (na rede de Petri de baixo nível), estruturação do algoritmo com dualidade de lugar-transição e, por fim, a localidade do efeito das transições. Portanto a rede de Petri é uma expressiva

linguagem gráfica de modelagem que permite a construção e validação do algoritmo de controle. Logo, mostra-se pertinente a implementação de um emulador de rede de Petri como forma de controle da bancada simulada. Tendo em vista que a bancada real é controlada por um CLP, este não possui interpretador direto da linguagem gráfica de rede de Petri. Logo, essa carência pode ser contornada pela implementação de um emulador.

Os benefícios da utilização do diagrama da rede de Petri para modelagem do controle de sistemas automatizados são evidenciados em (OSMAN; BAKAR, 2014). O estudo mostra uma análise da modelagem de um emulador de rede de Petri em comparação com a aplicação de um Diagrama Ladder, uma linguagem padronizada e comumente usada no campo dos controladores lógicos programáveis (PLCs). Estas duas linguagens gráficas foram aplicadas em um processo discreto de uma planta industrial composta por um tanque, como um exemplo para papel de pesquisa. Foram citadas vantagens da utilização do diagrama de rede de Petri, como a existência de um suporte matemático altamente desenvolvido e de alta flexibilidade na análise de todos os elementos que podem influenciar na condução de um evento. Também na descrição do sistema modelado graficamente de modo mais claro e intuitivo permitindo uma fácil visualização do sistema complexo. Ademais, a modelagem de sistemas de forma hierárquica possibilita a apresentação deste de cima para baixo em vários níveis de abstração e detalhamento. Por fim, possibilita a análise sistemática e qualitativa do sistema por meio de técnicas de análise de rede de Petri bem desenvolvidas. O texto conclui exprimindo que à medida que os sistemas de manufatura automatizados se tornam mais complexos, a necessidade de uma ferramenta de projeto eficaz para produzir sistemas de controle de eventos discretos de alto nível e implementações de baixo nível torna-se mais importante e que a rede de Petri representa o método mais eficaz tanto para o projeto quanto para a implementação destes sistemas.

Outro trabalho no qual é discorrido a aplicabilidade da rede de Petri para a simulação e validação de sistemas discretos, fica claro a benesse da utilização desta linguagem. Em (QUEZADA et al., 2017) o autor propõe a simulação e validação de algoritmos de controle desenvolvidos em Diagrama Ladder utilizando Redes de Petri para lidar com possíveis situações de falha (curto-circuito e / ou circuito aberto) no subsistema de entradas físicas de um sistema de controle baseado em CLP, por meio do caso de uso de um sietema de controle automático de lavagem de carro. Em virtude da rede de Petri possuir ferramentas para realização de análise do sistema modelado, o método proposto, baseado no emprego da rede de Petri a partir do Diagrama Ladder, mostrou-se adequado. A proposta de validação permitiu avaliar o

comportamento do algoritmo de controle nas possíveis condições de falha nos sinais físicos de entrada dos sensores, a fim de determinar condições de risco ou perigo que podem ocorrer no processo industrial, e portanto, permitindo tomada de medidas de segurança adequadas antes de sua implementação, ou mesmo se estas já estiverem implementadas nos sistemas baseados em CLP.

Ainda nessa conjuntura, (OSMAN; BAKAR, 2014) ressaltam a significância do emprego da teoria da rede de Petri no contexto atual, visto que sua aplicação aos sistema de manufatura comporta uma pesquisa muito importante para desenvolver um sistema de controle de manufatura moderno. A sua utilização para modelagem de sistema dinâmico de eventos discretos, planejamento e projeto de controle, encontra-se cada vez mais em ênfase. Portanto o desenvolvimento e aplicação de um emulador de rede de Petri neste trabalho, mostra-se pertinente.

2.2 Revisão de tecnologia

Tendo em mente o contexto, tecnologias e aplicações do Gêmeo Digital contemplado pelos estudos levantados anteriormente, foram pesquisados possíveis tecnologias que permitam a implementação do projeto. Primeiramente são discutidos *softwares* que propiciam a construção e implementação do Gêmeo Digital de bancada MPS Festo.

Há a existência de alguns *softwares* da Festo que estão relacionados com a simulação de processos e tecnologias da Indústria 4.0. O *software* COSIMIR proporciona uma ferramenta para simulações 3D, planejamento de células de trabalho baseadas em robôs, desenvolvimento de programas para robôs e controladores (RESEARCH, 2000). Existe também um *software* mais atual e flexível chamado CIROS, o qual possibilita a criação de simulações 3D de automação de processos e equipamento no âmbito fabril, simulação de processos da vida real e comportamento de componentes, permite a criação de aplicações de robótica e programação e teste de CLP (FESTO, 2021).

Apesar da disponibilidade dos *softwares* da FESTO e uma maior facilidade da simulação das bancadas MPS da FESTO nestes programas, optou-se por utilizar o *software* Visual Components, devido a sua alta flexibilidade, mais opções e versatilidade de simulação de processos, sua grande otimização e versão atualizada. Além do mais, possui APIs (*Application Programming Interface*) que adicionam mais funcionalidades e possibilidades, como o Python API e OPC-UA API que se mostram bastante úteis para a aplicação deste trabalho.

Discorrendo um pouco mais sobre o protocolo OPC-UA, é um protocolo de comunicação máquina a máquina para automação industrial. É citado em (OPC-UA, 2021) que "A Arquitetura Unificada OPC (UA), lançada em 2008, é uma arquitetura orientada a serviços independente de plataforma que integra todas as funcionalidades das especificações OPC Classic individuais em uma estrutura extensível". É o protocolo padrão utilizado no contexto da Indústria 4.0 e é suportado por diversos *softwares* de simulação. Por seu caráter universal, apresenta benefícios significativos quanto a escalabilidade de software e traz aos projetos replicabilidade além de oferecer "as melhores condições de digitalização, independente de plataforma e fabricante e com mecanismos de segurança integrados". (SIEMENS, 2021). O que se mostra a viabilidade de sua aplicação de forma a estabelecer a comunicação entre o Gêmeo Digital da bancada MPS Festo e o emulador de rede de Petri.

Agora sobre o desenvolvimento do emulador, dentre as linguagens de programação possíveis, a linguagem Python apresenta-se como a opção mais estimada devido à sua sintaxe objetiva e enxuta, alta flexibilidade e grande variedade de bibliotecas facilitando e possibilitando diversas aplicações. No caso, Python possui uma biblioteca do protocolo OPC-UA que contribui para a implementação deste no emulador, permitindo-se o estabelecimento da comunicação do emulador com a bancada MPS simulada.

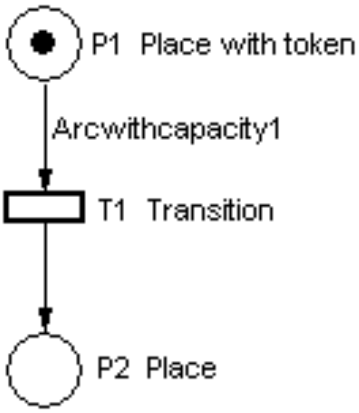
Em síntese será utilizado o *software* de desenvolvimento de simulações, Visual Components, para projetar, desenvolver e implementar o Gêmeo Digital de uma bancada MPS da Festo. Os API's de Python do Visual Components serão utilizados para comandar o comportamento da simulação e o protocolo OPC-UA será responsável por estabelecer a comunicação entre um interpretador de rede de Petri desenvolvido em Python e o Visual Components, permitindo o controle da simulação por um arquivo XML de rede de Petri.

2.3 Rede de Petri

A rede de Petri é uma técnica de modelagem que permite a descrição e análise de sistemas discretos. Foi originalmente formulada por Carl Adam Petri, que dá nome à técnica, e desenvolvida a partir do trabalho inicial de Petri sobre o tema, datado de 1962, quando formulou a base da teoria de comunicação entre componentes assíncronos de um sistema computacional. Essa técnica de representação de processos destaca-se das demais representações de fluxos visuais ou matemáticas para sistemas distribuídos discretos ao permitir modelar sistemas paralelos, concorrentes e assíncronos se forma simples e bastante objetiva.

Nesse sentido, a rede de Petri, enquanto forma de descrição do procedimento de controle de sistemas e eventos discretos, sua gráfica é formada por dois principais componentes, um ativo, as transições, e um passivo, os lugares. Tais componentes são representados respectivamente pelas formas de retângulos e círculos, respectivamente. Os lugares, enquanto componentes passivos, podem assumir estados, capazes de armazenar ou mostrar itens. Nesse sentido, podem estar vazios ou cheios, e isso será representado visualmente na rede de Petri da figura 1.

Figura 1 – Representação de rede de Petri simplificada com 2 lugares e uma transição intermediária.



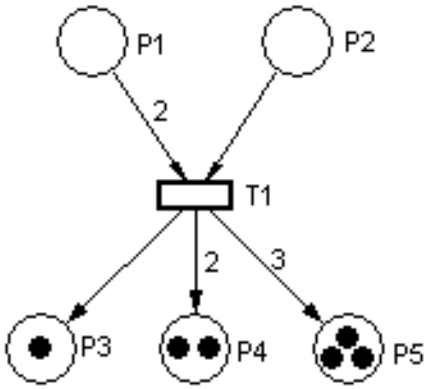
Fonte: (CHEN, 2002)

Na rede de Petri, os componentes representados na figura 1 são ligados entre si por arcos dirigidos, que, por sua vez, podem ser únicos ou múltiplos. Diferentes dos ativos e distribuidores, os arcos não representam um componente do sistema, mas sim um relacionamento abstrato entre os componentes (MIYAGI, 1996). Os arcos são representados por linhas, que denotam uma direcionalidade através de setas em uma das pontas da linha. Logo, os arcos podem ter duas direcionalidades, sendo orientado de transição ao lugar ou de lugar a transição. Não é possível, entretanto, que o arco seja orientado de lugar a lugar ou de transição a transição. Caso isto ocorra durante a criacao de uma rede de Petri, diz-se que o modelo assumido esta omitindo um nivel de informação.

Por definição, os lugares têm capacidade infinita. Caso contrário, a capacidade é indicada no próprio Lugar.

Um tipo especial de arco, o arco inibidor, é usado para reverter a lógica de um local de entrada. Com um arco inibidor, a ausência de uma Marca no local de entrada permite, não a presença. Além disso, um Arco pode conter um peso específico que, por padrão é 1. Caso um

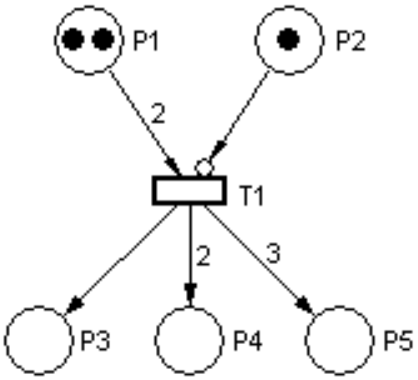
Figura 2 – Representação de rede de Petri: Lugares com capacidades distintas.



Fonte: (CHEN, 2002)

Arco com peso 2, por exemplo, saia de um Lugar, ele exigirá 2 Marcas no Lugar de origem para que a Transição ao qual aponta seja ativada. Caso um Arco com peso 3, por exemplo, saia de uma Transição, ele alocará 3 Marcas no Lugar de destino após ativada a Transição de origem.

Figura 3 – Representação de rede de Petri: Arco inibidor.



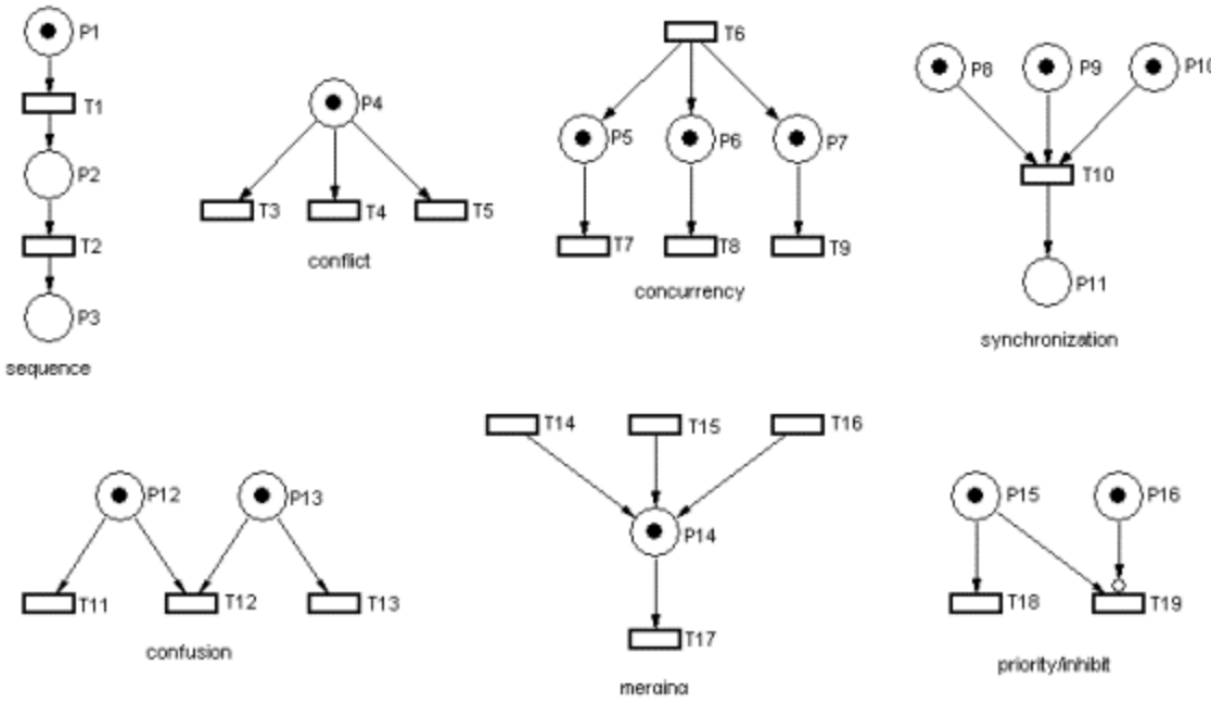
Fonte: (CHEN, 2002)

Na figura 4, pode-se verificar algumas das estruturas chamadas de primitivas no contexto de redes de Petri. A partir de tais estruturas é possível elaborar redes de Petri complexas para atender as mais diversas cadeias de eventos.

2.4 Bancada MPS

O termo MPS refere-se a *Modular Production System*, chamadas de Sistema Modular de Produção (SMP) em portugues, são sistemas criados pela companhia FESTO, utilizados como ferramentas de treinamento e cursos de programação para técnicos da área de automação

Figura 4 – Representação de redes de Petri: Primitivas.



Fonte: (CHEN, 2002)

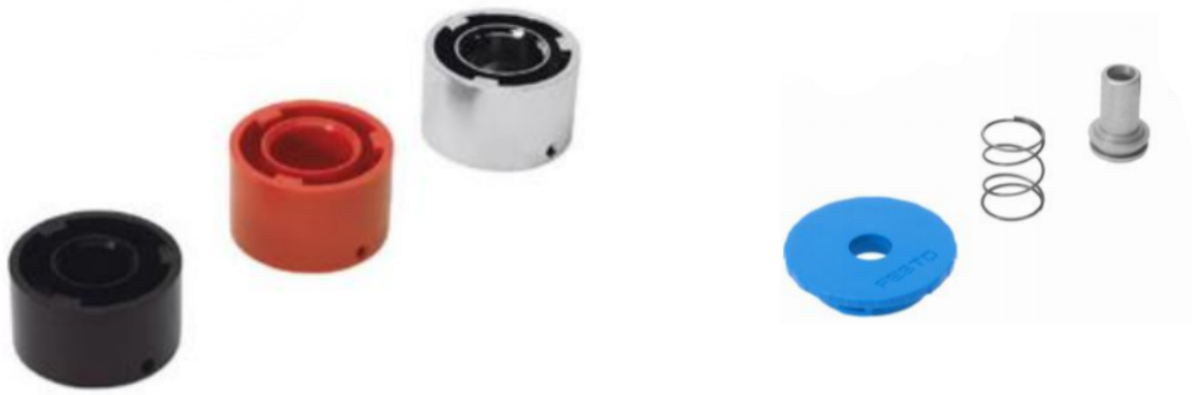
industrial. Foi concebida para a formação prática e é modelada com a máxima relevância industrial em automatização e tecnologia de manipulação de peças. Devido às ferramentas e recursos de aprendizagem disponíveis no sistema, pode-se criar um ambiente de aprendizagem ideal para a formação em mecatrônica e para a automatização fabril. Além do mais, foi projetado de forma a valorizar a formação independente, básica e superior relacionada à indústria na mecatrônica e na tecnologia de automatização, como também o valor a longo prazo e a robustez do equipamento.

Devido ao alto nível de modularidade do sistema, pode combinar estações, módulos e acessórios para criar uma linha de produção personalizada de acordo com os objetivos e cenários de aprendizagem. As estações são plataformas de produção individuais, as quais realizam funções específicas no processo. Estas são compostas por módulos os quais correspondem aos componentes e instrumentos que combinados permitem a realização da função específica de produção. Os acessórios são dispositivos que permitem estender e controlar os MPS.

As bancadas MPS atuam em peças de trabalho padronizadas e desenvolvidas especificamente para esses sistemas. Existem dois conjuntos de peças de trabalho, cada conjunto possui sua peça base, a qual pode ser manipulada em qualquer das estações, e peças complementares, as quais podem ser montadas nas peças bases em certas estações.

O primeiro conjunto de peças, corresponde à peça base, tipo camisa de cilindro, juntamente com as peças complementares, pistão, mola e tampa. Quando montada a peça base com as complementares, obtém-se pequenos cilindros pneumáticos. As peças base são cilindros com espaços internos os quais contemplam três cores possíveis: vermelho, preto e prata, visíveis na figura 5, sendo os de cores vermelho e preto constituídos de plástico e o prata de metal. O diâmetro do cilindro é padrão em 40mm e pode possuir três alturas diferentes, sendo 22,5 mm, 25 mm e 27,5 mm. As peças complementares correspondem a tampa em cor azul, a mola e o pistão que dispõe de duas cores (preto e prata). O pistão preto possui altura de 20mm e acomoda apenas as peças vermelhas e pratas. Já o pistão prata possui altura de 16mm e acomoda apenas as peças pretas. Como adendo, apenas as peças base de altura 22,5 mm e 25 mm podem ser montadas com as complementares.

Figura 5 – (a) Peças base tipo corpo de cilindro e (b) peças complementares.



Fonte: (POLA, 2013)

As estações MPS possuem uma estrutura comum configurada pela planta (onde ficam instalados os módulos), o painel de controle (interface homem máquina para controle manual das estações), o gabinete móvel (armação da estação) e um controlador lógico programável. Alguns módulos comuns também estão presentes nas plataformas. O módulo de unidade de conservação é utilizado para regular a pressão e filtrar o ar comprimido da estação. O módulo de sensor óptico receptor e transmissor, que respectivamente, recebe sinal óptico digital de um sensor transmissor, quando a estação subsequente está pronta para receber a próxima peça (sinal em 0 define a bancada livre, enquanto sinal em 1 define a bancada como ocupada) e transmite um sinal óptico digital da mesma maneira.

Ademais, discorre-se sobre as estações MPS. As principais estações MPS disponíveis

são nomeadas de acordo com a função específica que realizam e são listadas a seguir com uma breve descrição:

MPS-Distributing, essa estação é responsável por fornecer as peças de trabalho ao sistema. Pode-se armazenar até nove peças em um módulo de depósito, atuado por um cilindro pneumático de dupla ação para saída da peça. O módulo de transferência de peças é composto por um atuador giratório com uma ventosa para a sucção das peças.

MPS-Testing, essa estação recebe peças, realiza a verificação da peça e determina se a peça será aprovada, prosseguindo para uma próxima estação, ou rejeitada, isolando a peça na própria plataforma, de acordo com os critérios estabelecidos. Por meio de sensores, é possível realizar a classificação da peça de acordo com sua cor e em sequência verifica-se a altura da peça.

MPS-Processing, essa estação recebe peças, transporta as peças de trabalho por meio de uma mesa giratória, e as processa. É realizada a indexação da peça para verificação da posição do orifício, em seguida, é realizada uma simulação de usinagem no orifício da peça. Após esse processamento a peça é encaminhada à próxima estação. Essa estação não possui nenhum atuador pneumático.

MPS-Sorting, essa estação recebe peças, realiza a classificação delas, e as separa por tipo ou por cor. As peças são detectadas no início da esteira e levadas até o módulo de parada, pelo qual é classificada pelos sensores ópticos e indutivo. Após a classificação, as peças são separadas nas três rampas, de acordo com os critérios estabelecidos na programação.

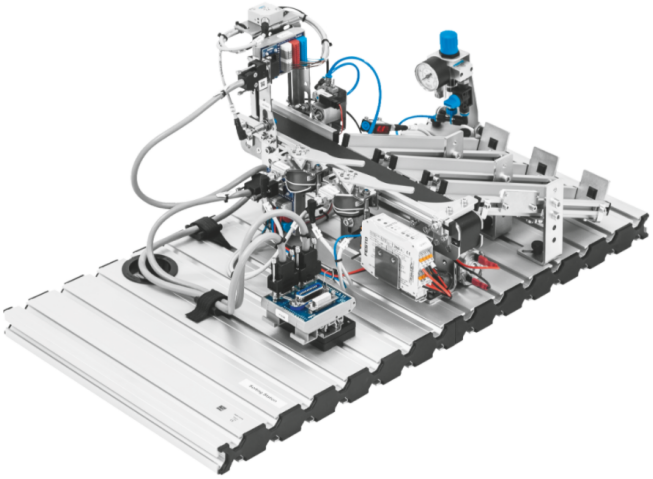
MPS-Handling Pneumática, essa estação é utilizada como estação de manipulação de peças. Ela recebe peças e pode também classificá-las, diferenciando as pelas cores, e realizando a separação das peças. A manipulação é realizada por meio de um manipulador, com uma garra pneumática específica para essa atividade.

Diante da exposição das principais estações MPS descritas acima, optou-se pela escolha da estação *MPS-Sorting*, disponível na figura 6, para prosseguir com o desenvolvimento do Gêmeo Digital desse sistema.

O objetivo desta plataforma, como foi citado anteriormente, é a classificação e separação das peças de trabalho pela cor ou pelo tipo da peça. Ademais, discorre-se sobre os módulos e instrumentos característicos desta estação. A figura 7 comporta os principais elementos da estação *Sorting*.

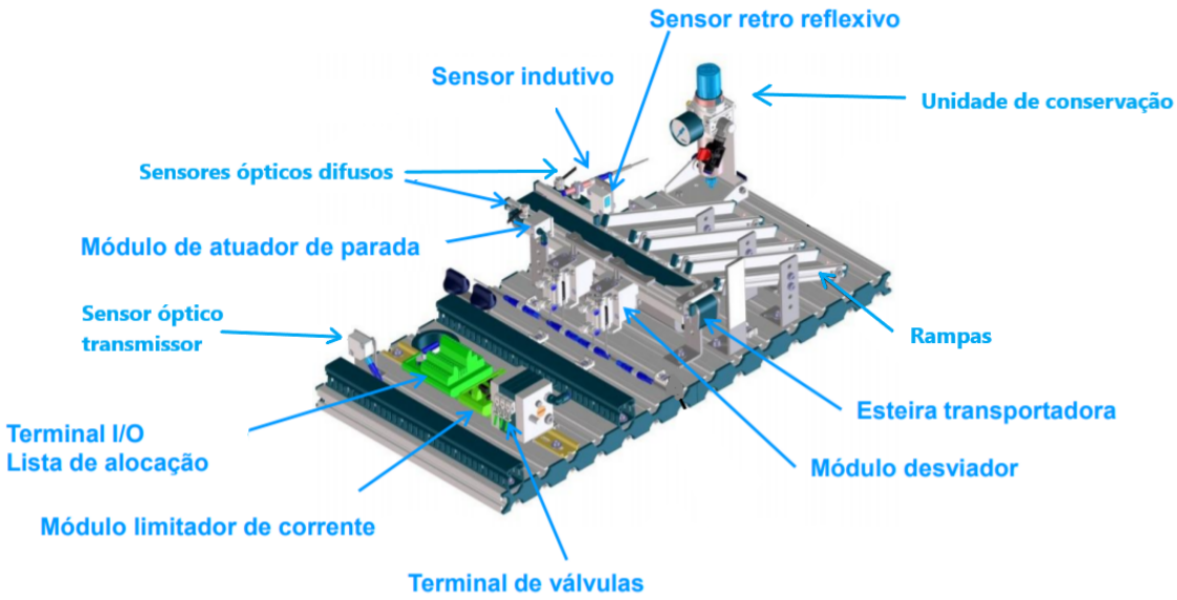
O fluxo da plataforma inicia-se com a inserção de uma peça de trabalho no início da esteira transportadora onde um dos sensores ópticos difusos detecta a peça de trabalho

Figura 6 – *Estação MPS-Sorting.*



Fonte: (FESTO, 2015)

Figura 7 – *Estação MPS-Sorting com indicação de seus principais elementos.*



Fonte: (FESTO, 2015)

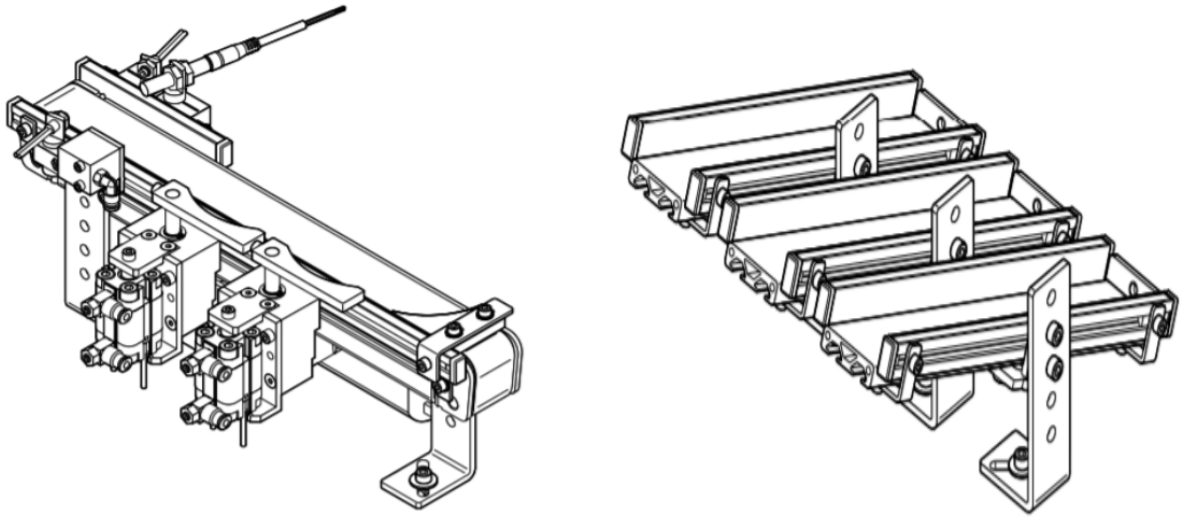
e marca o início da execução do código de controle. As peças de trabalho devem proceder individualmente para não prejudicar as funções de manobra dos ramos. Em seguida, a esteira transportadora é acionada e a peça fica barrada pelo módulo de atuador de parada. Então a peça é classificada de acordo com a cor e o tipo pelo outro sensor óptico difuso e o sensor indutivo, respectivamente. Após a classificação finalizada, o módulo desviador definido para a respectiva peça é acionado, caso a peça não seja alocada para a última rampa, e o módulo de atuador de parada recua o pistão permitindo a passagem da peça na esteira. O sensor retro

reflexivo monitora o nível de enchimento das rampas e se alguma rampa atinge sua capacidade máxima, o módulo atuador de parada não recua o pistão, mesmo depois de classificar a peça no início do processo.

Ademais, discorre-se sobre os módulos que não atuam diretamente à peça de trabalho, mas possuem funções necessárias para o funcionamento dessa estação. Os módulos de sensor óptico de transmissão e unidade de conservação são comuns às estações, como já explicado anteriormente. O módulo limitador de corrente é um dispositivo utilizado como proteção no acionamento do motor da esteira e possui um botão para o acionamento manual do motor. O módulo do terminal de válvulas é utilizado para o acionamento dos dois desviadores e do módulo de parada. Já o módulo de terminal *I/O* é usado para ligar oito entradas e oito saídas digitais, as quais são conectadas a soquetes, também há LEDs instalados nos terminais de entrada e saída, o que facilita o monitoramento do status de comutação.

Em síntese, a estação *Sorting*, possui duas partes principais compostas por módulos que interagem com a peça de trabalho, a parte transportadora e a parte das rampas de separação. A figura 8 ilustra em maior detalhe essas partes.

Figura 8 – (a) Transportadora e (b) rampas de separação.



Fonte: (FESTO, 2015)

2.5 Síntese

O presente capítulo elucidou o contexto da aplicação de Gêmeos Digital como ferramentas e tecnologias pertinentes para o desenvolvimento no sistema de ensino. Além do mais, evidenciou os benefícios da utilização da linguagem matemática e gráfica de rede de Petri como forma de descrever o controle do processo baseado em eventos discretos. Também houve uma descrição mais detalhada da estrutura e funcionamento de redes de Petri, como forma de estabelecer uma base teórica para melhor entendimento do funcionamento do emulador, o qual terá seu projeto descrito nos próximos capítulos, que servirá de controle do Gêmeo Digital. Além disso, o capítulo discorreu em mais detalhes sobre o que são as bancadas MPS Festo, dando um panorama e breve descrição das principais estações que compreendem o sistema MPS, e apresentou em detalhes o funcionamento da estação escolhida para desenvolvimento do projeto de Gêmeo Digital presente no próximo capítulo, como forma de estabelecer uma base para a composição e construção da forma digital desta plataforma.

3 Metodologia do Projeto

3.1 Projeto do Gêmeo Digital da Bancada MPS

Este capítulo tem como intuito descrever o panorama da construção, estruturação e comportamento do Gêmeo Digital da bancada *MPS-Sorting* pelo *software* de desenvolvimento de simulações industriais, *Visual Components* (Visual Components, 2021) e descrever a metodologia de implementação, do programa de controle a partir de leitura e interpretação de arquivo XML. Inicialmente é retratado de modo geral as etapas sequenciais determinadas para execução do projeto. Em seguida, aborda-se em mais detalhes o que cada etapa compreende, bem como o procedimento e as características que ela implementa. O detalhamento das etapas de estruturação do Gêmeo Digital é baseado nos componentes da plataforma *MPS-Sorting*, descritos no capítulo anterior na seção 2.4, enquanto o detalhamento das etapas de desenvolvimento do Programa é baseado em metodologia UML.

Para uma melhor compreensão do projeto e do sistema do Gêmeo Digital, mostra-se pertinente expor como se dá a representação dos componentes no *Visual Components*. Existem três elementos básicos que fazem parte de um componente: *Features*, *Behaviors* e *Properties*.

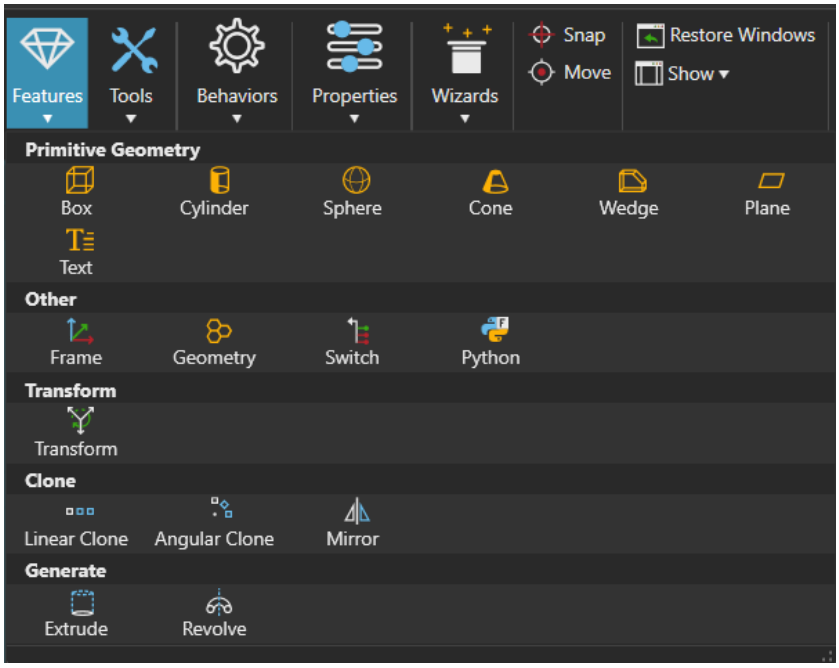
As *Features* caracterizam a representação visual de um componente. Elas são usadas para conter, agrupar e editar a geometria, bem como definir pontos de referência em um componente. Os *Behaviors* definem comportamentos e executam tarefas em um componente. As *Properties* são variáveis globais em um componente. As figuras 9, 10, e 11, mostram, respectivamente, os tipos e opções de *Features*, *Behaviors* e *Properties*, que podem ser implementados na criação de um componente.

A estrutura de um componente é definida por uma árvore determinada por uma hierarquia de nós. Cada nó contém um conjunto de *Behaviors* e *Features*. O nó raiz é a origem de um componente e contém o conjunto de *Properties* deste. Cada nó vinculado ao nó raiz tem seu próprio deslocamento e junta, que define a amplitude de movimento e os graus de liberdade desse nó. *Behaviors* em nós diferentes podem ser conectados e referenciados uns aos outros. Os *Features* em nós formam uma hierarquia e podem ser aninhados uns com os outros para realizar operações que manipulam a geometria de um recurso e do próprio objeto.

A Figura 12 sintetiza as etapas do projeto do Gêmeo Digital:

A primeira etapa compreende a definição dos componentes, a qual é responsável pela

Figura 9 – *Painel de Features.*

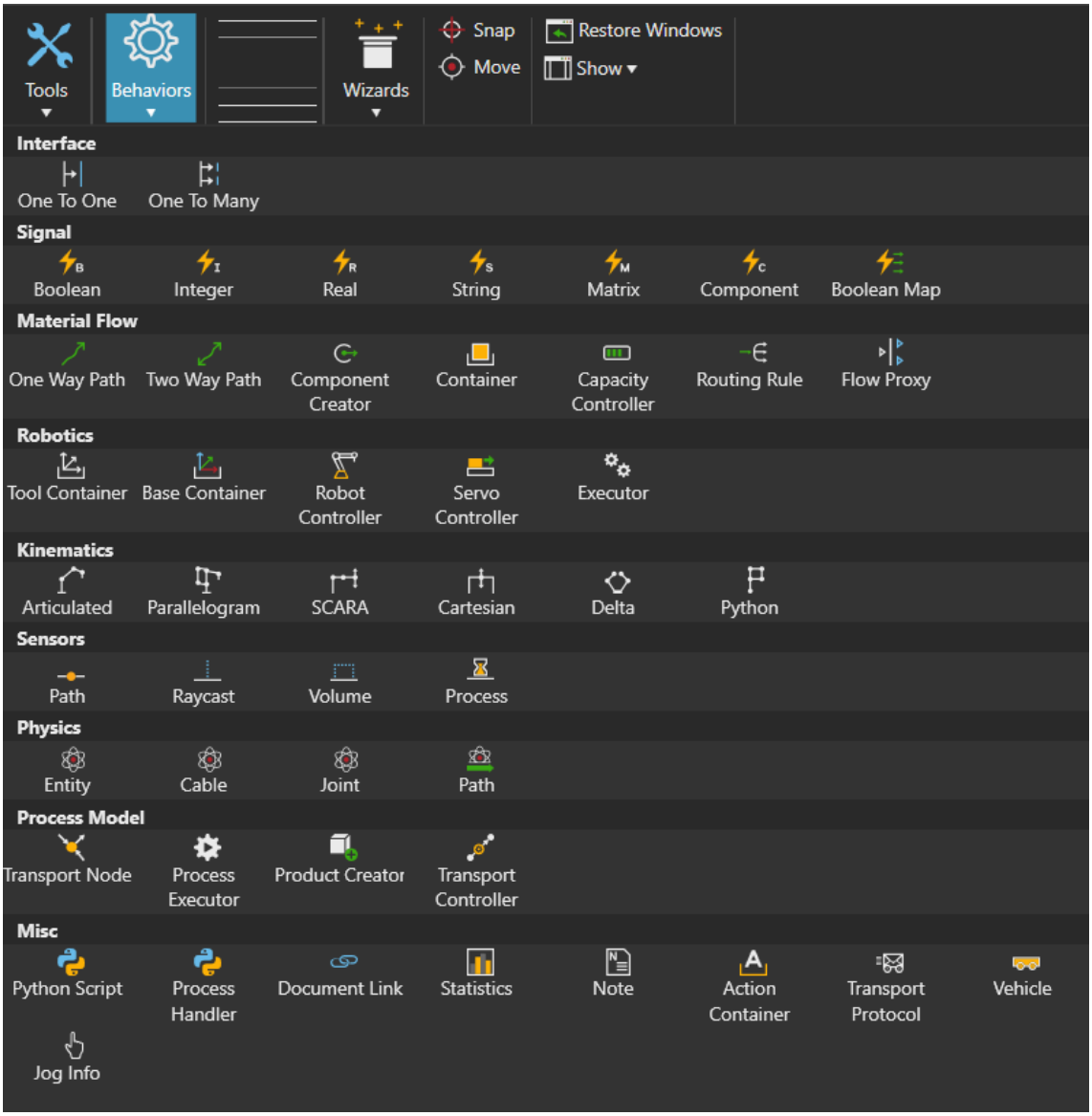


Fonte: (Visual Components, 2021)

importação das geometrias as quais estabelecem as *Features* de cada elemento que compõem a bancada em questão. Essas *Features* serão reorganizadas de modo a determinar como será a composição de cada componente dentro do *Visual Components*, assim como a as juntas de certos componentes que possuem alguma função na manipulação da peça de trabalho.

A segunda etapa estabelece as características de cada componente. Isso estende-se no âmbito da definição das *Behaviors* e *Properties*. A implementação das *Properties* será aplicada para adicionar variáveis globais a certos componentes, as quais serão vinculadas ao *script* que será implementado na terceira etapa e também, para definir as restrições, velocidades e acelerações das juntas dos componentes atuadores. Ademais, serão implementadas restrições físicas das juntas dos componentes atuadores. Já a implementação de *Behaviors*, refere-se: ao estabelecimento das interfaces de cada componente, isto é, a conexão dos componentes com fluxo da peça de trabalho e de sinais entre componentes; à definição das rotas nos componentes que serão responsáveis pelo fluxo da peça de trabalho; a implementação de sensores que serão vinculados a certos sinais; ao estabelecimento dos sinais auxiliares necessários para o funcionamento da simulação, bem como os sinais de entrada e saída que serão trocados com o emulador, depois da implementação da última etapa de conectividade OPC-UA; por fim, à definição de servo controladores os quais serão vinculados às juntas dos atuadores, permitindo a simulação do movimentos destes.

Figura 10 – *Painel de Behaviors.*



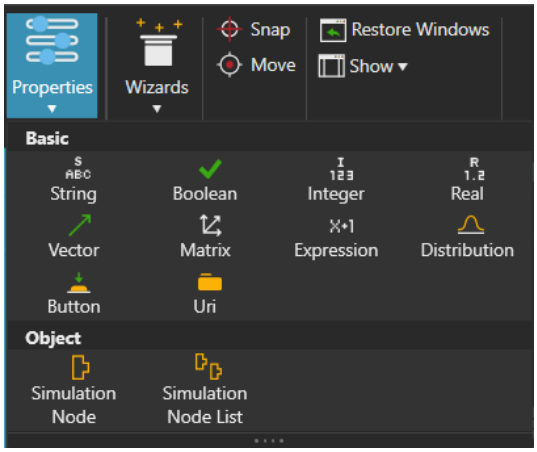
Fonte: (Visual Components, 2021)

A terceira etapa compreende a implementação de um *Behaviors*, fundamental para o funcionamento da simulação, o *Python Script*, disponibilizado pelo API Python do *Visual Components*. Este será responsável pela aplicação da lógica de funcionamento e da tratativa de sinais do Gêmeo Digital durante a simulação e troca dos sinais com o emulador. Será aplicado em apenas um componente central, facilitando a elaboração do algoritmo.

A quarta e última etapa refere-se à conexão entre o Gêmeo Digital e o emulador de rede de Petri, a partir do protocolo OPC-UA.

A seguir, discorre-se em mais detalhes cada etapa do projeto do Gêmeo Digital.

Figura 11 – *Painel de Properties.*



Fonte: (Visual Components, 2021)

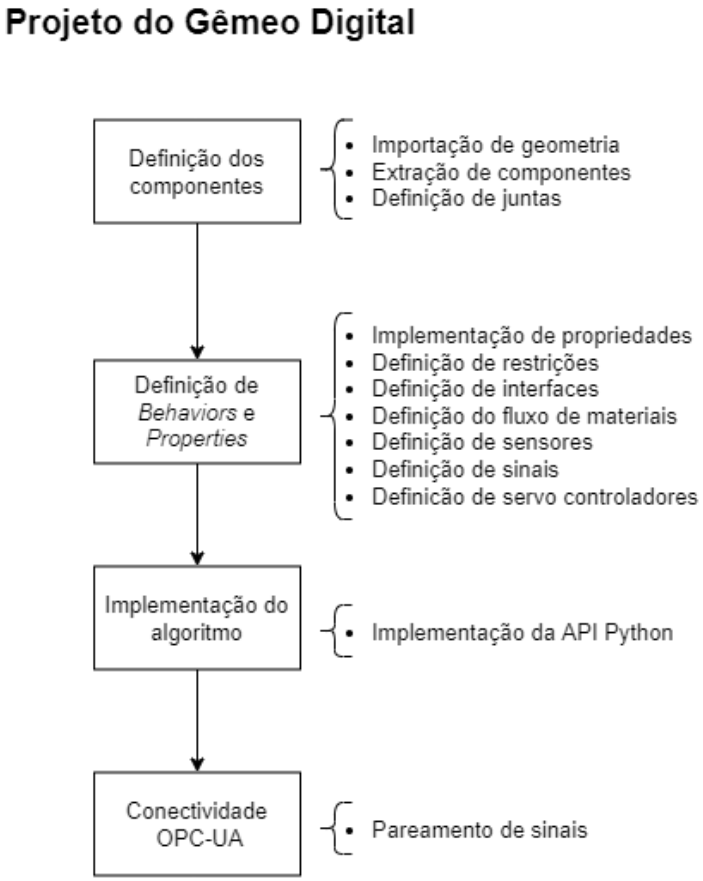
3.1.1 Definição dos componentes

Inicialmente, para definição dos componentes, é necessário importar a geometria das peças. Para tal, será utilizado o CAD da bancada *MPS-Sorting*, disponibilizado pela própria Festo para o laboratório de mecatrônica da Universidade de São Paulo. O *Visual Components* importa as geometrias advindas de um arquivo CAD, de modo a formar um único componente composto de diversas *Features*. Logo se faz necessário extrair os componentes a partir dessa estrutura inicial de forma a organizar e separar as *Features* para formar os componentes individualmente. Além disso, para os componentes que possuem fluxo da peça de trabalho, faz-se necessário adicionar *Features* de coordenadas as quais são chamadas de *Frames*, para com isso, posteriormente definir os *Behaviors* de rotas do fluxo das peças de trabalho e da posição de sensores.

Partindo-se dos 15 componentes principais da plataforma *MPS-Sorting* apresentados na seção 2.4, é descrito a seguir quais comporão um único componente e quais terão juntas. Os componentes são: planta, painel de controle, gabinete móvel, terminal de válvulas, módulo limitador de corrente, terminal *I/O* (*input*/*output*), sensor óptico receptor, unidade de conservação, módulo atuador de paradas, módulos desviadores, esteira transportadora, rampas de separação, sensores ópticos difusos, sensor indutivo e sensor retro reflexivo.

O primeiro componente será definido pela composição dos módulos que não interagem fisicamente ou sensorialmente com as peças de trabalho. Este será composto pelos módulos: planta, painel de controle, gabinete móvel, terminal de válvulas, módulo limitador de corrente, terminal *I/O*, sensor óptico receptor, unidade de conservação. O componente formado será

Figura 12 – *Projeto do Gêmeo Digital proposto*



apenas ilustrativo e servirá de base para os demais componentes.

O segundo componente será composto pela esteira transportadora, pelos dois sensores ópticos difusos, pelo sensor indutivo e pelo sensor retro reflexivo. Visto que, como será descrito posteriormente, os locais onde os sinais serão captados estarão definidos pela posição destes sensores na esteira transportadora e nas rampas separadoras.

As rampas separadoras darão origem a 3 componentes individuais, sendo cada componente uma das rampas separadoras. Os dois módulos desviadores e o módulo atuador de parada compõem individualmente o próprio componente e possuem juntas. O componente atuador de parada possui uma junta prismática, a qual translada um cilindro para simular o bloqueio da passagem da peça de trabalho. Os dois componentes desviadores possuem uma junta rotativa que permite a rotação de um braço desviador, simulando o redirecionamento do curso da peça de trabalho para alguma das rampas separadoras.

Faz-se importante discorrer sobre as peças de trabalho processadas na bancada. O modelo da peça de trabalho é extraído do CAD da bancada fornecido. A partir desse modelo

são determinados três componentes que retratam as três peças base tipo corpo de cilindro, ilustradas na Figura 8.

Além da definição dos componentes extraídos do CAD importado, será utilizado um componente auxiliar disponibilizado pelo *Visual Components* chamado de *Advanced Feeder* o qual terá a função de gerar as peças de trabalho que entrarão na bancada durante a simulação.

3.1.2 Definição de *Behaviors* e *Properties*

A definição das *Properties* se aplicará com a incorporação de uma variável global para cada um dos três componentes que definem as peças de trabalho. A função dessa propriedade adicionada será a de conter a informação sobre a cor do componente. Estas variáveis serão utilizadas, posteriormente, no *Python Script*. Ademais, é necessário definir as restrições e velocidades dos componentes atuadores do processo, estes são: os dois módulos desviadores e o módulo atuador de parada. Para os módulos desviadores serão definidas propriedades referentes à junta de rotação, sendo esta restrita a uma rotação entre 0 e 55 graus, com a velocidade e aceleração mantida a padrão, respectivamente, de $100°/s$ e de $500°/s^2$. Já o componente de módulo atuador de parada terá a propriedade de translação da junta prismática com um deslocamento mínimo de 0 mm e máximo de 13 mm, com a velocidade e aceleração mantida a padrão, respectivamente, de $100\text{mm}/s$ e de $500\text{mm}/s^2$.

Sobre a definição de *Behaviors*, inicialmente será definido o interfaceamento. Este compreende a conexão física entre os componentes que comportam o fluxo da peça de trabalho e a conexão de sinais entre componentes. As interfaces definirão fluxo de entrada e fluxo de saída da peça, utilizando os *Frames* definidos anteriormente. O componente correspondente à esteira transportadora terá uma interface de entrada para fluxo de material e três interfaces de saída de fluxo de material, também terá mais três interfaces de sinais. Estas interfaces serão conectadas respectivamente com o a interface de saída de fluxo de material do *Advanced Feeder*, com cada interface de entrada de fluxo de material das três rampas separadoras e com cada uma das três interfaces de sinais de cada rampa separadora.

Posteriormente, serão definidas rotas de fluxo da peça de trabalho sobre alguns componentes. Estes são: a esteira transportadora e cada um dos três componentes das rampas separadoras.

Ademais, será definido os *Behaviors* do tipo "sensores". Estes *Behaviors* tem como objetivo identificar a passagem da peça de trabalho nas rotas de fluxo de material e serão

utilizados de forma análoga aos sensores reais da bancada. Serão utilizados 5 sensores do tipo *Raycast* que serão posicionados em certos *Frames*, identificados posteriormente. Esse tipo de sensor é acionado quando uma peça de trabalho corta o eixo z do *Frame* em que foi definido. Dois desses *Behaviors* estarão no componente da esteira transportadora, o primeiro será posicionado na direção do primeiro sensor óptico difuso e o segundo será posicionado na direção do segundo sensor óptico difuso e do sensor indutivo. Cada um dos outros três *Behaviors* estarão em uma das rampas separadores e sua função será análoga ao sensor retro-reflexivo.

Acerca da definição de sinais, estes serão divididos em dois grupos: os sinais de entrada e saída, que serão mapeados com os sinais do emulador de rede de Petri, e os sinais auxiliares, que serão utilizados apenas no próprio gêmeo digital. O primeiro grupo de sinais será baseado nos próprios *I/Os* da estação real, e cada um está relacionado com alguma funcionalidade da estação. A tabela 1 apresenta a relação de *I/Os* e as funcionalidades da estação *MPS-Sorting*.

Tabela 1 – Terminal de I/Os - Estação MPS Sorting.

Terminal de I/O Entradas digitais (IN)	**Descrição**	**Terminal de I/O Saídas digitais (OUT)**	**Descrição**
DI 0	Peça disponível no início da esteira	DO 0	Liga esteira
DI 1	Peça metálica	DO 1	Avança atuador 1
DI 2	Peça não preta	DO 2	Avança atuador 2
DI 3	Rampa cheia	DO 3	Recua atuador de parada
DI 4	Atuador 1 recuado	DO 4	
DI 5	Atuador 1 avançado	DO 5	
DI 6	Atuador 2 recuado	DO 6	
DI 7	Atuador 2 avançado	DO 7	Estação ocupada

Fonte: (POLA, 2013)

Nota-se que o terminal de *I/Os* descrito na tabela está com o referencial do controlador da estação, pois os sinais denominados com "*DI*" são os de entrada do controlador e saída do Gêmeo Digital e os sinais denominados com "*DO*" são os de saída do controlador e entrada do Gêmeo Digital. Optou-se por utilizar o mesmo nome dos sinais como apresentado na imagem para maior facilidade na identificação e, posteriormente, pareamento dos sinais entre o Gêmeo Digital e o emulador. Já com relação aos sinais auxiliares, estes serão definidos durante a implementação da simulação.

Por fim, serão definidos os *Behaviors* do tipo servo controladores os quais possibilitaram

o movimento dos componentes atuador linear, módulo desviador 1 e módulo desviador 2, durante a simulação.

3.1.3 Implementação do algoritmo

A terceira etapa do projeto será a implementação de um algoritmo que descreve a lógica básica de funcionamento do Gêmeo Digital, isto é, de acordo com as funcionalidade dos sinais 1, estabelece-se a lógica de mudança dos sinais de saída da simulação e executa-se as funções da estação de acordo com os sinais de entrada da simulação.

Este algoritmo é implementado pelo *Behavior* de *Python Script*, assim como foi citado anteriormente, será aplicado em apenas um componente central. Este componente será a esteira transportadora, o qual também compreenderá, além de outros *Behaviors*, o primeiro grupo de sinais definidos (de entrada e saída). O *script* utilizará alguns *Behaviors* e *Properties* criados anteriormente nos componentes para parte de seu funcionamento.

O algoritmo será baseado em duas funções principais. A primeira será responsável por verificar se há a mudança de algum sinal, e se houver, executa alguma outra ação. E a segunda será responsável pela simulação que será definida dentro de um *loop* constante e executará as funções de movimentação dos atuadores e do fluxo da peça de trabalho.

3.1.4 Conectividade OPC-UA

A quarta e última etapa corresponde ao mapeamento e pareamento dos sinais de entrada e saída definidos anteriormente, para com o os sinais do definido no emulador de rede de Petri, por meio do protocolo de comunicação OPC-UA. Utiliza-se o API OPC-UA disponível no *Visual Components*, proporcionando para o Gêmeo Digital a característica de conectividade, permitindo a comunicação com o emulador.

3.2 Projeto do Emulador de rede de Petri

O projeto do Emulador de rede de Petri tem como objetivo claro o recebimento de um arquivo XML (caracterizado por marcas relativas a uma rede de Petri) a ser processado e servir como estrutura para a implementação de um Sistema a Eventos Discretos capaz de controlar uma Máquina (neste caso uma representação visual) via protocolo OPC-UA.

Para uma rede de Petri como a disponível na Figura 13, o arquivo XML correspondente, desenvolvida a partir do *software* PIPE 4.3 (Platform Independent Petri net Editor, 2009) seria

tal qual o disponível na Figura 14.

Figura 13 – Exemplo de rede de Petri.

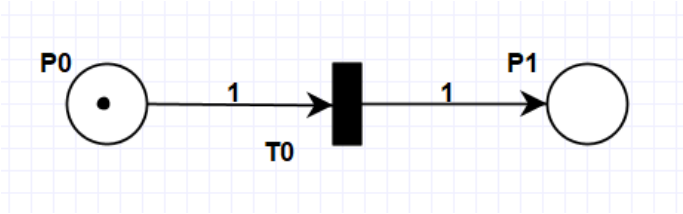


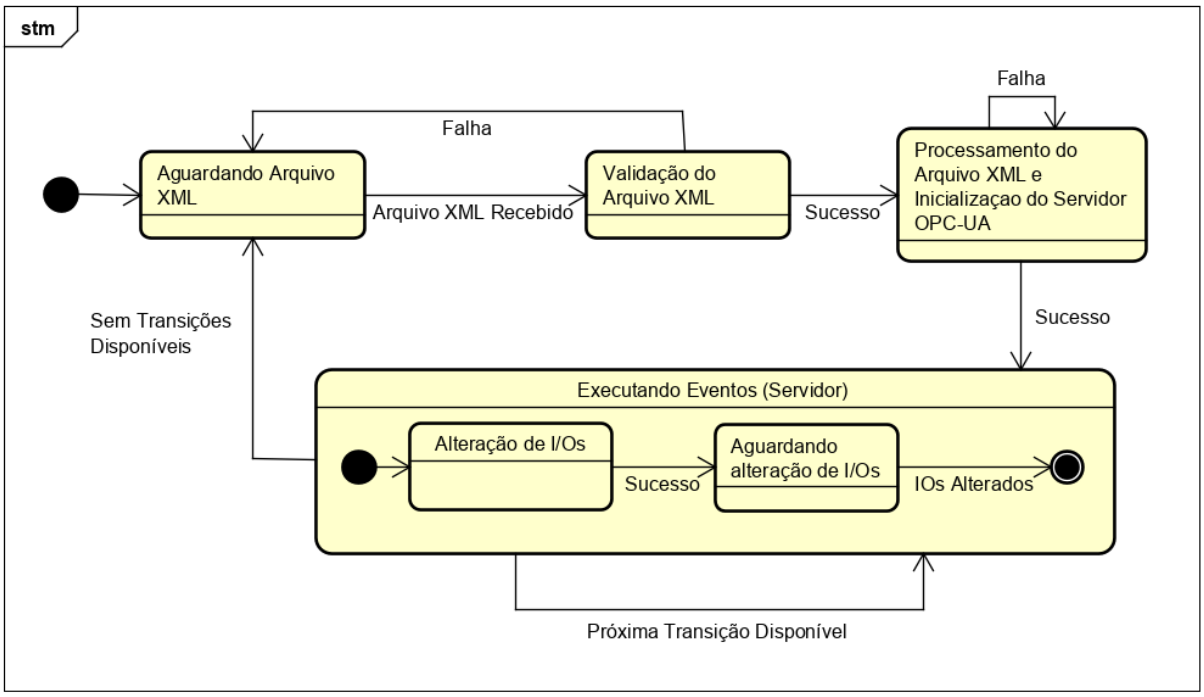
Figura 14 – XML (*Extensible Markup Language*) associado à Figura 13.

```
<?xml version="1.0" encoding="UTF-8"?>
<pnml>
   <net id="Net-One" type="P/T net">
      <token id="Default" enabled="true" red="0" green="0" blue="0" />
      <place id="P0">
         <graphics>···
         </graphics>
         <name>···
         </name>
         <initialMarking>
            <value>Default,1</value>
            <graphics>···
            </graphics>
         </initialMarking>
         <capacity>
            <value>0</value>
         </capacity>
      </place>
      <place id="P1">···
      </place>
      <transition id="T0">···
      </transition>
      <arc id="P0 to T0" source="P0" target="T0">···
      </arc>
      <arc id="T0 to P1" source="T0" target="P1">···
      </arc>
   </net>
</pnml>
```

Para a implementação do Emulador de rede de Petri, utilizou-se a linguagem de programação Python. Inicialmente foram revisados os objetivos do programa com base na metodologia UML.

O diagrama da Máquina de Estados, encontrado na Figura 15, tem por objetivo representar os possíveis caminhos que o programa pode percorrer. Dado que existe uma comunicação assíncrona entre o emulador de rede de Petri (que representa justamente um sistema de eventos discretos) e o gêmeo digital (que por sua vez representa um sistema contínuo) faz-se necessária a criação de estados de espera, cujas transições estão representadas no diagrama de Máquina de Estados.

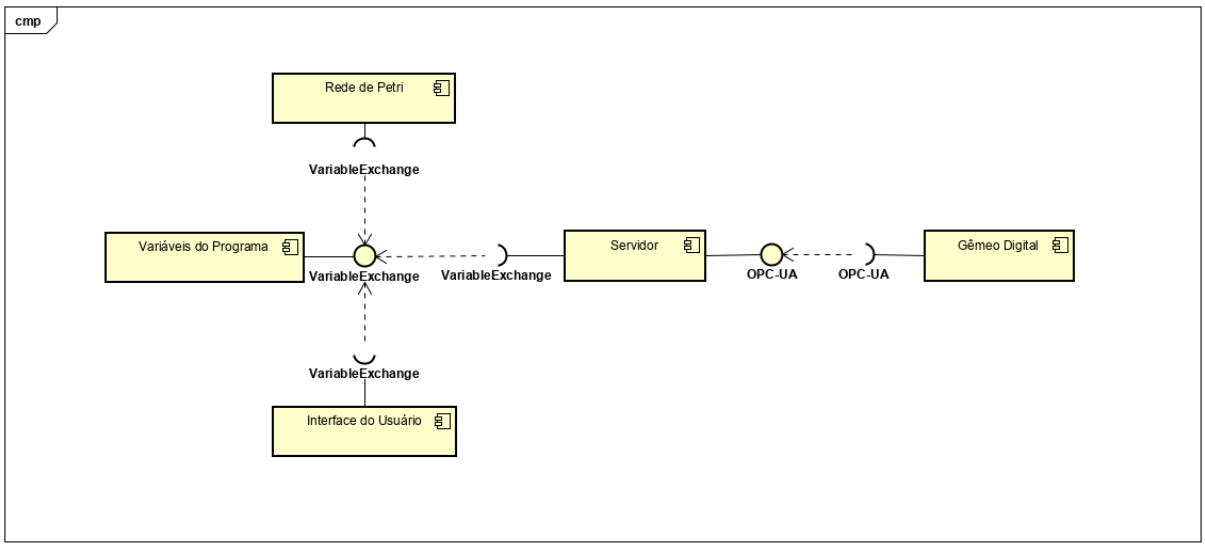
Figura 15 – *Diagrama de Máquina de Estados UML.*



Neste caso, é esperada, conforme Figura 15 uma etapa em que o programa aguarda a leitura correta de um arquivo XML, para que então ocorra o processamento do arquivo, a inicialização do Servidor OPC-UA, seguida por fim de um ciclo de alteração e leitura dos *I/Os*.

Por sua vez, o diagrama de Componentes, encontrado na Figura 16, foi elaborado a fim de estrututar o funcionamento e a interação das frentes do programa, bem como representar o protocolo OPC-UA que faz interface entre o emulador e o Gêmeo Digital.

Figura 16 – *Components Diagram UML.*



Para a implementação de todas as três frentes propostas (interface do usuário, interpretador de rede de Petri e servidor OPC-UA), foram elencadas algumas dependências de bibliotecas em Python a serem utilizadas, dentre elas:

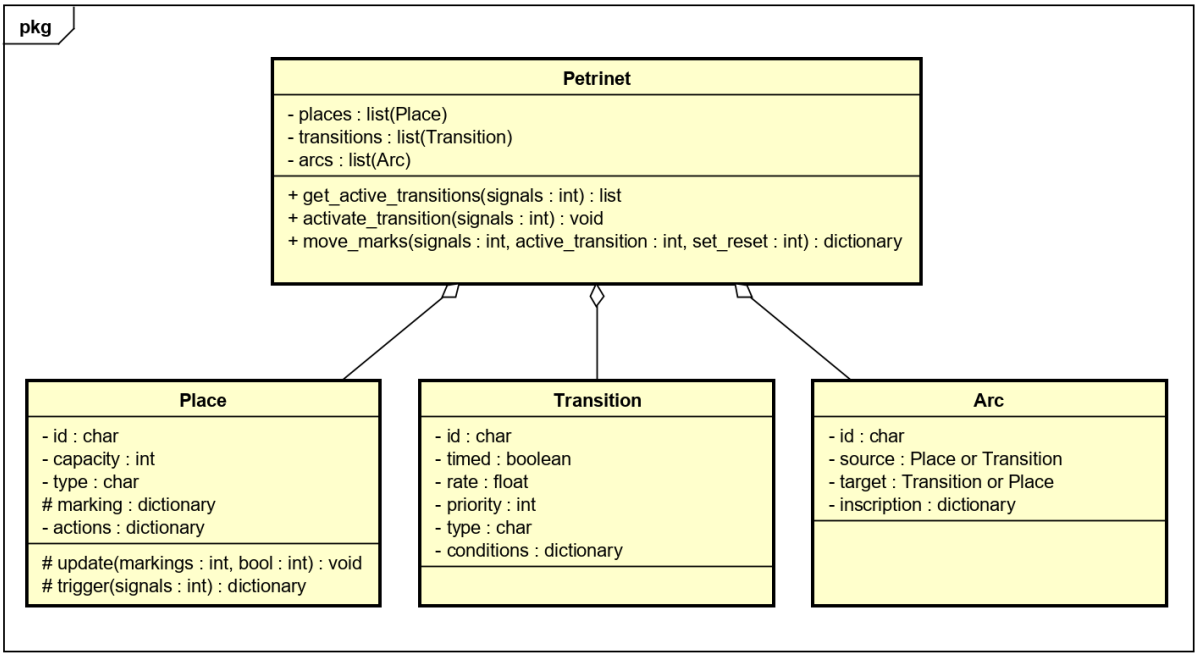
PySimpleGUI para a implementação de uma Interface para o Usuário de forma simplificada. A interface tem como proposta a definição de parâmetros essenciais para o estabelecimento do protocolo OPC-UA bem como os arquivos de entrada do usuário e de variáveis de controle do programa, além de possibilitar ao usuário a escolha de um arquivo compatível XML.

freeopcua para a implementação de um servidor OPC-UA, permitindo a troca de informação entre o emulador de rede de Petri e o Gêmeo Digital.

xml para a interpretação hierárquica (*parse*) do arquivo XML inserido pelo usuário.

A estrutura da rede de Petri é, por sua vez, implementada a partir de Classes e Objetos desenvolvidos no programa. Os lugares e transições são genericamente definidos de modo a serem flexibilizados de acordo com um arquivo de entrada do usuário contendo a descrição da rede de Petri. A rede de Petri por sua vez é inicializada conforme estado inicial definido pelo arquivo inserido sendo estruturada de acordo com o Diagrama de Classes UML disponível na Figura 17.

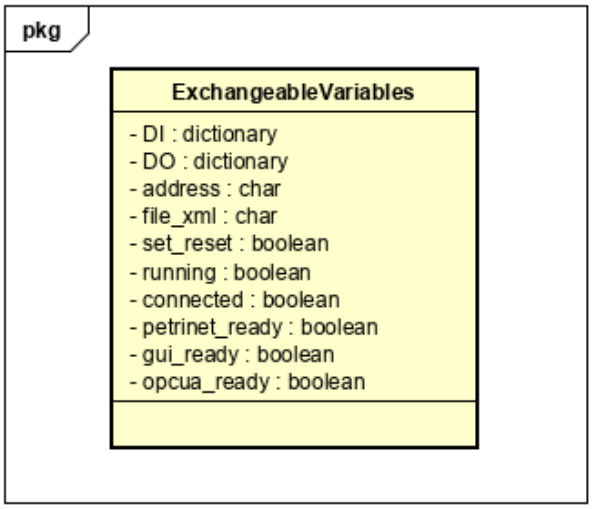
Figura 17 – *Emulador de rede de Petri - Diagrama de Classe UML.*



Por fim, conforme proposto pelo diagrama de Componentes da Figura 16, estabeleceu-

se mais um Diagrama de Classes UML, disponível na Figura 18 a fim de estruturar as variáveis do programa que devem ser consultadas e atualizadas ao longo de toda a execução do mesmo. Além das informações disponíveis na Figura 18, é na Classe *ExchangeableVariables* onde são armazenadas constantes do programa, tais como cores da interface e textos de introdução a serem apresentados na interface do usuário.

Figura 18 – *Variáveis do Programa - Diagrama de Classe UML.*



4 Desenvolvimento

Este capítulo documenta a implementação de ambas as partes do trabalho, do Gêmeo Digital da bancada *MPS-Sorting* e do emulador de rede de Petri, baseando-se na estrutura de projeto definida nos capítulos anteriores. Além do mais, o capítulo contempla a integração das duas partes por meio do protocolo de comunicação máquina a máquina, OPC-UA. Uma descrição mais detalhada do funcionamento geral do processo é posta. Ela engloba uma explicação do mapeamento de sinais de ambas as partes e da correlação e fluxo da lógica básica estabelecida na estrutura dos projetos.

4.1 Gêmeo Digital da Bancada MPS

A implementação do desta seção seguiu a estrutura do fluxograma do projeto do Gêmeo Digital da bancada *MPS-Sorting* da Figura 12. A primeira parte de execução do projeto foi a definição dos componentes, a segunda parte foi a definição de *Behaviors* e *Properties*, a terceira parte foi a implementação do algoritmo e por fim, o estabelecimento da conectividade OPC-UA.

4.1.1 Definição dos componentes

Iniciou-se a construção do Gêmeo Digital com a importação da geometria da bancada *MPS-Sorting* por meio do arquivo CAD fornecido pelo fabricante. A Figura 19 expõe o componente gerado com a importação da geometria.

Após a importação foi necessário redefinir a base de coordenadas do componente. Foi posta na roda esquerda posterior da estrutura do gabinete móvel. Com a importação do arquivo CAD, foi gerado um único componente que corresponde a todos os elementos da bancada *MPS-Sorting*, definido por apenas uma *Feature* como é ilustrado na Figura 20.

Nota-se no canto inferior esquerdo na janela de *Component Graph*, a árvore de *Features* do componente em questão que existe apenas a *Feature "Collapsed0"*. O painel *Component Graph* fornece um esboço do componente selecionado, incluindo seus nós, *Behaviors*, *Properties* e *Features*. Para que seja possível definir diferentes componentes para, posteriormente, definir os seus *Behaviors* e *Properties*, foi preciso derivar essa única *Feature* em diversas outras que descrevem a geometria de cada forma individual de cada elemento da bancada *MPS-Sorting*.

Figura 19 – *CAD importado.*

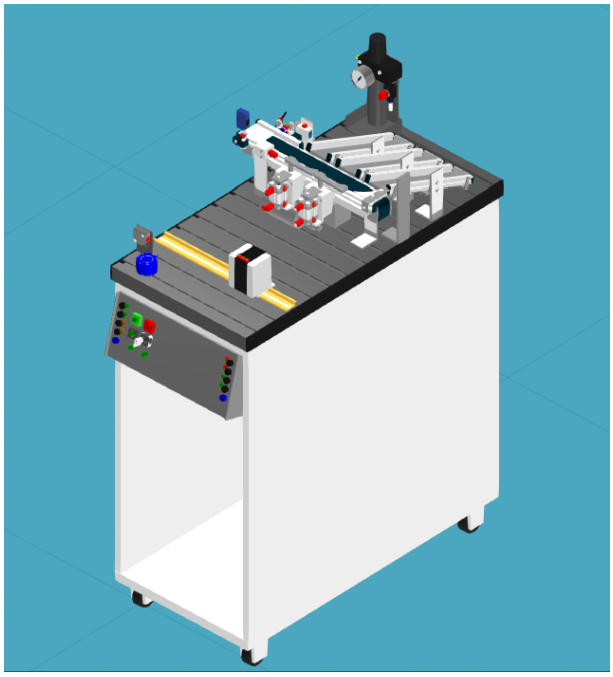
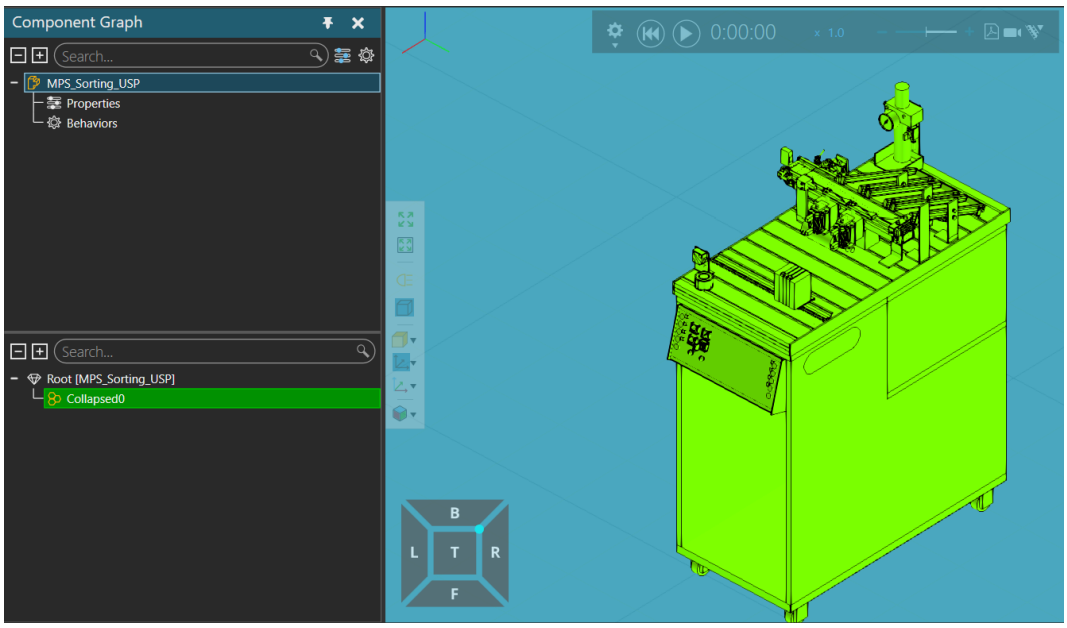


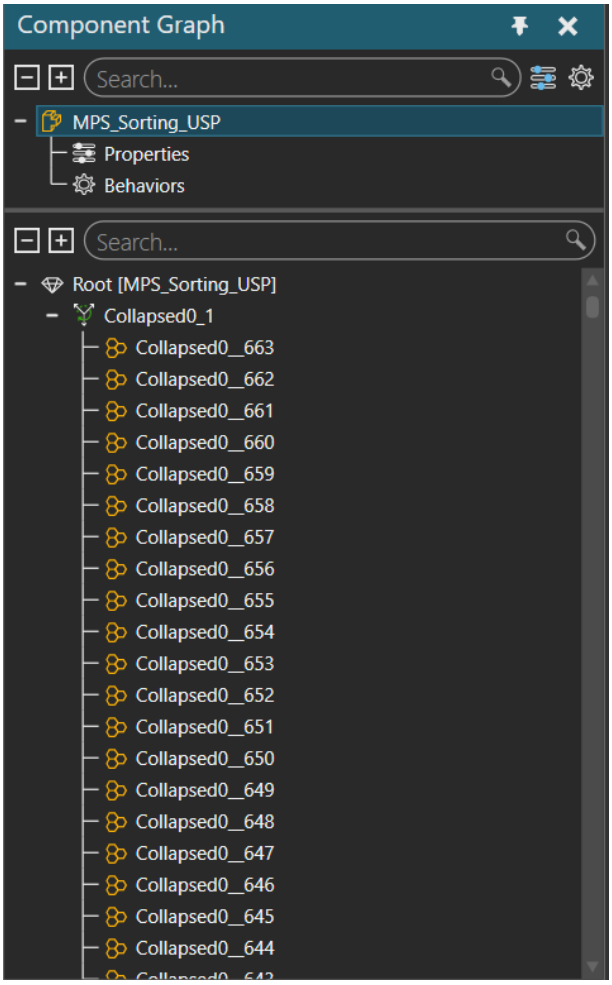
Figura 20 – *Componente gerado pela importação do CAD.*



Para isso, utilizou-se o comando de "*Explode*", gerando 663 *features*. A árvore de *features* ficou conforme a Figura 21:

Tendo cada elemento e geometria definida por uma *Feature*, iniciou-se o processo de seleção destas para extraí-las como um novo componente. Para todo novo componente gerado, foi definido um sistema de coordenadas para o próprio. A ordem de definição dos componentes foi baseada no número de *features* que cada componente compreende, os primeiros foram os com menos *features*. Assim, facilitou-se para definir o último componente, que é definido pela

Figura 21 – *Component Graph da bancada importada após a aplicação do "Explode".*



maioria das *features*, o qual corresponde a todos os elementos da bancada *MPS-Sorting* que não interagem diretamente com a peça de trabalho. O primeiro componente extraído foi a peça de trabalho.

A Figura 22, ilustra o componente genérico gerado para a peça de trabalho tipo corpo de cilindro. A partir desse componente genérico, foram geradas três cópias desse arquivo com o objetivo de definir as três diferentes peças de trabalho, estabelecendo os materiais e cores das respectivas peças na segunda etapa de definição de *Behaviors/Properties*.

O segundo componente extraído foi o módulo atuador de parada, denominado como atuador linear. Uma das *Features* que formaram o componente foi extraída como uma junta e as demais foram fundidas em apenas uma *Feature*, para melhor organização da estrutura da geometria do componente. A junta definida corresponde ao cilindro que executa o movimento de translação linear, responsável por simular o bloqueio da passagem da peça de trabalho na esteira transportadora. O componente do atuador linear e seu *Component Graph* podem ser observados na Figura 23.

Figura 22 – *Componente genérico da peça de trabalho.*

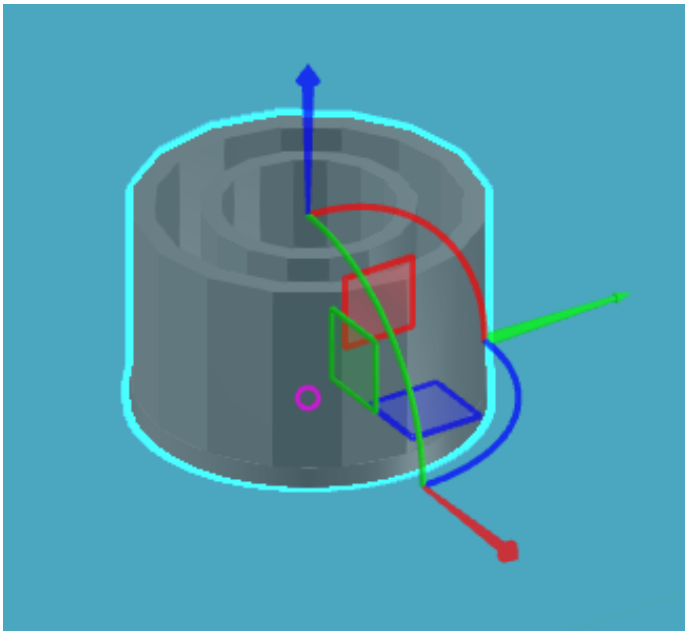
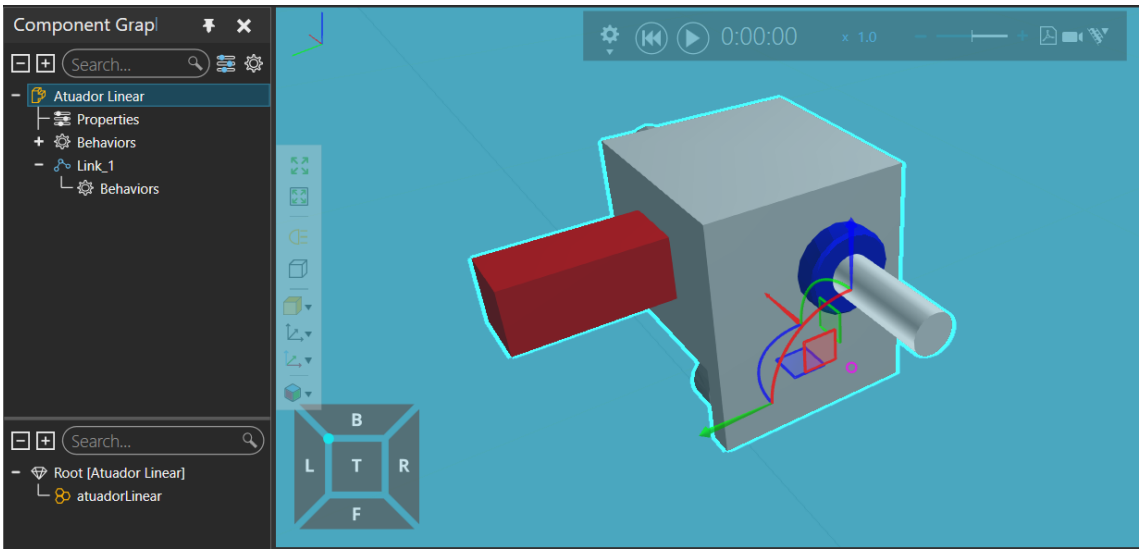


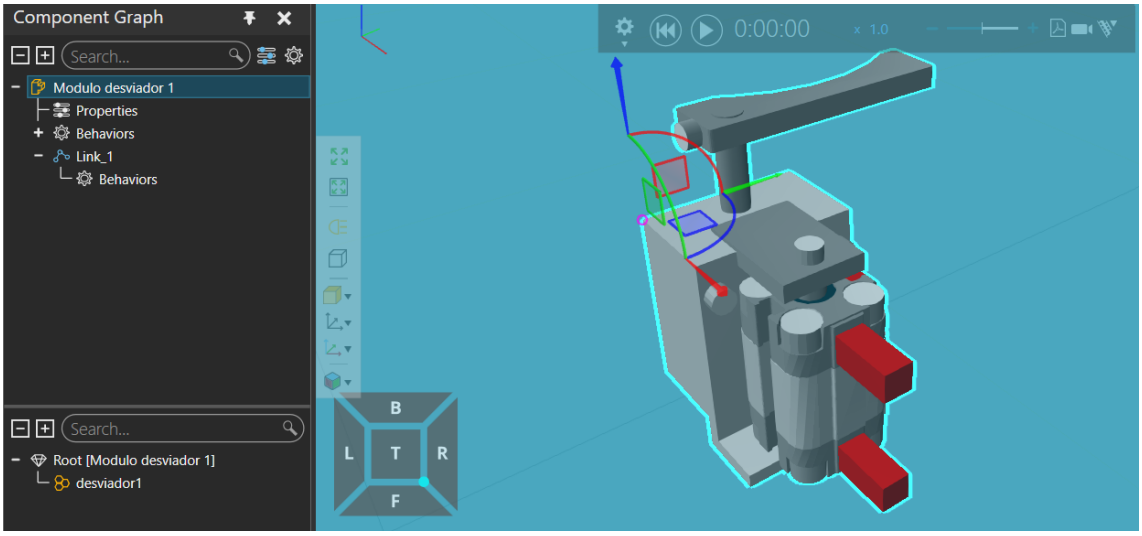
Figura 23 – *Módulo atuador de parada e seu Component Graph.*



O terceiro e quarto componentes representam os dois módulos desviadores e foram definidos de forma igual. Para cada um foi definida uma junta que executa o movimento de rotação, responsável por simular o desvio da peça de trabalho para as rampas separadoras. As *Features* que compunham a junta foram fundidas em uma única *Feature* e as que compunham o resto do componente foram fundidas em uma outra. A Figura 24 exibe o componente e seu *Component Graph*:

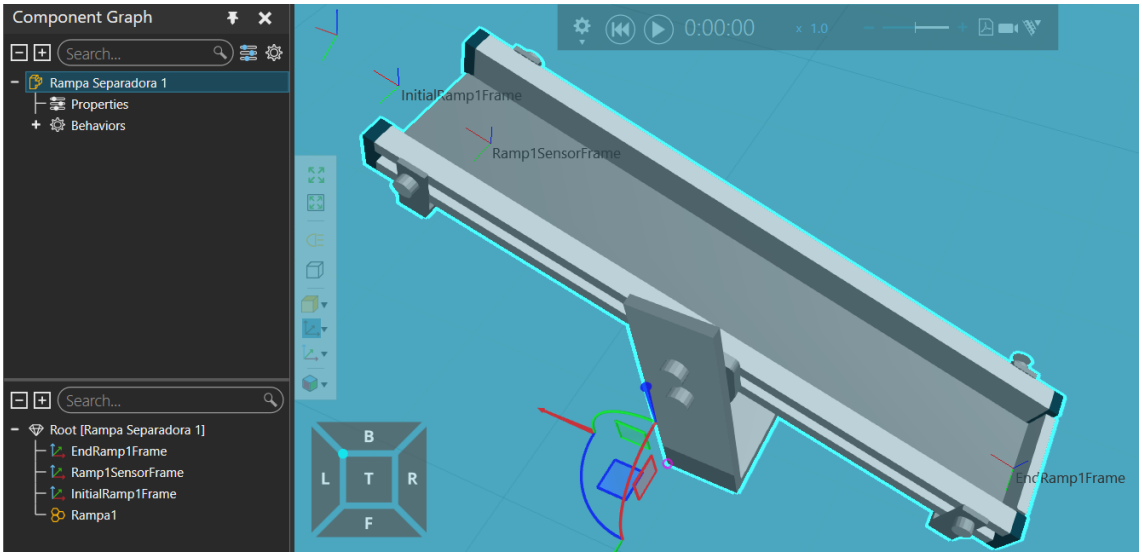
O quinto, sexto e sétimo componentes definidos foram as três rampas separadoras. Da mesma forma, para cada rampa, suas *Features* foram fundidas em apenas uma. Além disso, para cada rampa foi adicionado um conjunto de *Frames*, os quais serão utilizados posteriormente

Figura 24 – *Módulo desviador e seu Component Graph.*



para definir *Behaviors* do tipo interface, sensores e rotas de fluxo de material. A Figura 25 exibe o componente de uma rampa e seu *Component Graph*:

Figura 25 – *Rampa separadora e seu Component Graph.*



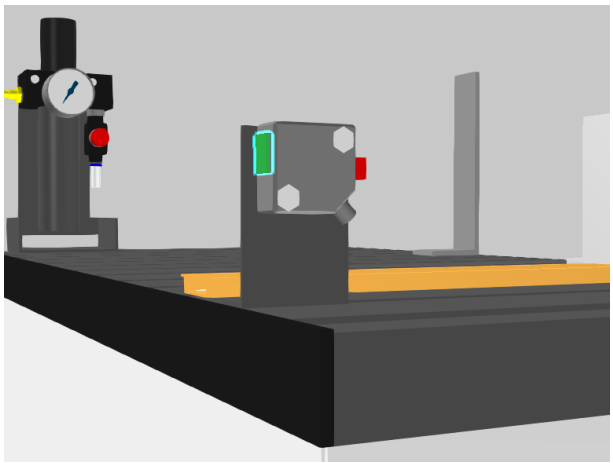
O oitavo componente definido foi o módulo da esteira transportadora, composta pela esteira transportadora, pelos dois sensores ópticos difusos, pelo sensor indutivo e pelo sensor retro reflexivo. Também foi definido para esse componente um conjunto de *Frames* que serão utilizados posteriormente para definir *Behaviors* do tipo interface, sensores e rotas de fluxo de material. Do mesmo modo que os componentes definidos anteriormente, suas *Features* foram fundidas em apenas uma. A Figura 26 representa este componente, assim como seu *Component Graph*.

Posteriormente, foi extraído um componente adicional, a partir de uma das *Features*

Figura 26 – *Esteira transportadora e seu Component Graph.*

restantes. Este componente compreende um *Led* localizado no sensor óptico receptor e foi criado com o objetivo de indicar se a estação está ocupada ou não. A ideia é que sua cor mude para verde se a estação estiver livre ou mude para vermelho se estiver ocupada, essa mudança da cor do material do componente foi implementada no *Python Script* posteriormente. A localização desse componente na estação é ilustrada na Figura 27.

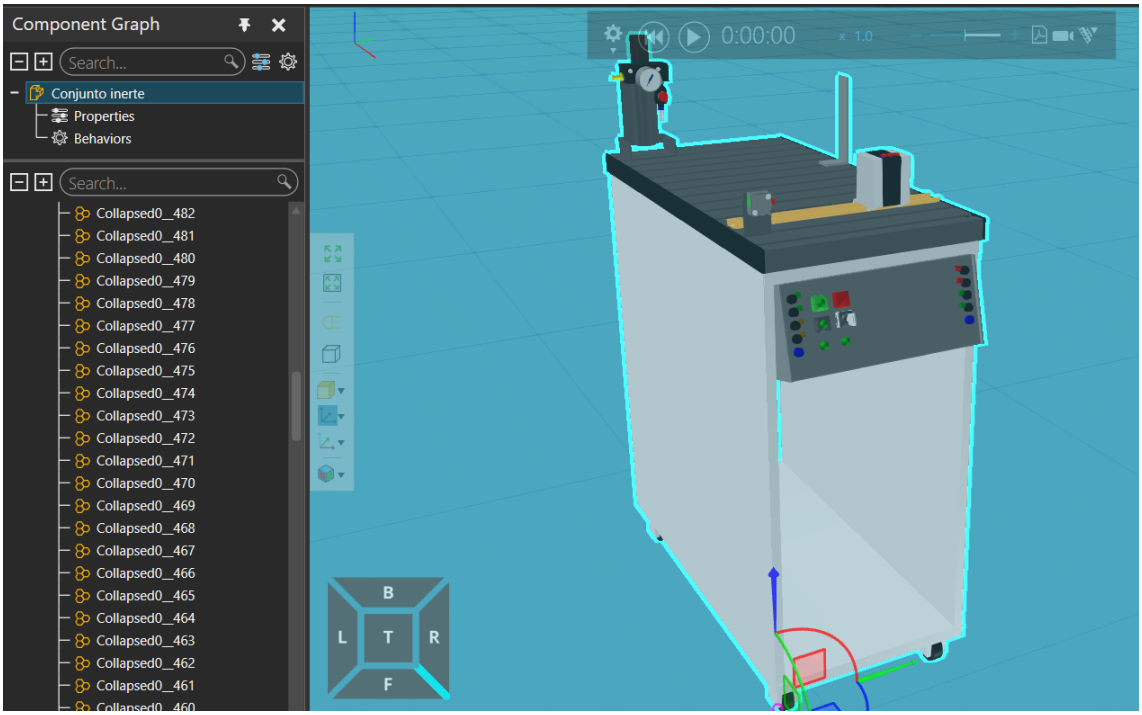
Figura 27 – *Componente Led.*



O décimo componente foi nomeado como "Conjunto Inerte" e foi definido com as *Features* restantes após a definição dos componentes anteriormente, estabelecendo a composição dos módulos que não interagem fisicamente ou sensorialmente com as peças de trabalho. Neste caso, suas *Features* foram deixadas de forma individual, pois caso houvesse uma necessidade de criar um outro componente a partir destes elementos, bastaria extrair as *Features* corres-

pondentes como um novo componente. A Figura 28 ilustra esse componente e seu *Component Graph*.

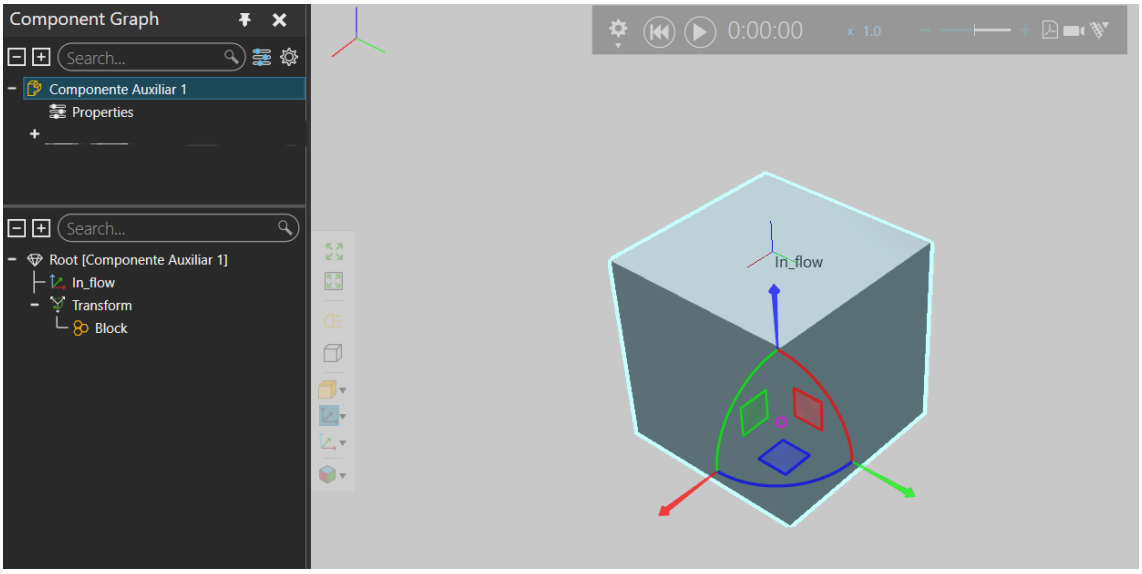
Figura 28 – *Conjunto Inerte e seu Component Graph.*



Além do mais, implementou-se mais três componentes, denominados como componentes auxiliares. Estes foram implementados com o objetivo de adicionar uma funcionalidade extra para a simulação, a de acúmulo de peças nas rampas separadoras. A ideia por trás desses componentes foi de agirem como esteiras transportadoras com capacidade zero, as quais seriam conectadas ao final das rampas separadoras por meio de interfaces. Assim quando estas conexões estivessem ligadas, as peças de trabalho acumulariam na rampa, e quando estivessem desligadas, as peças de trabalho desapareceriam da simulação. Será abordado mais à frente essa lógica nas etapas de definição de *Behaviors/Properties* e na de implementação do algoritmo. De todo modo, para criação desse componente utilizou-se uma estrutura básica de um cubo disponível no próprio *software* e foi adicionado um *Frame*, o qual seria utilizado para definir os *Behaviors* tipo interface e rota de fluxo de material. Nota-se que esses componentes permaneceram com a opção de visibilidade desativada, já que não fazem parte fisicamente da estação mas apenas executam uma funcionalidade para uma simulação verossímil. A Figura 29 mostra um desses componentes com a opção de visibilidade ativada apenas para efeito de documentação.

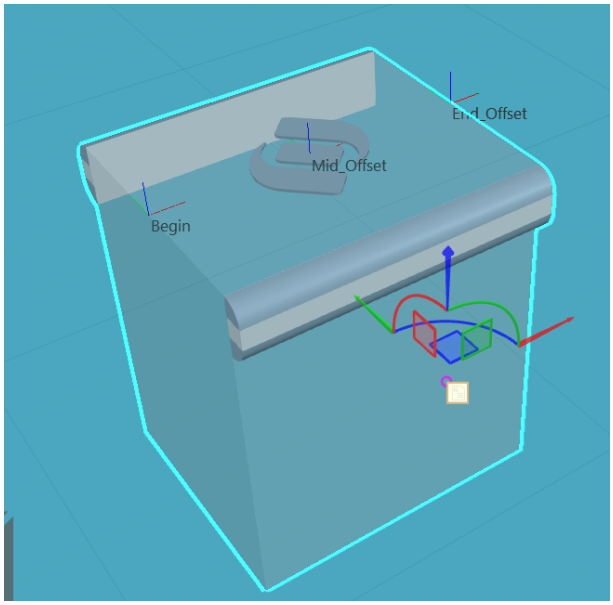
Por fim, foi adicionado ao Gêmeo Digital o componente abordado anteriormente, o

Figura 29 – *Componente auxiliar e seu Component Graph.*



Advanced Feeder. A Figura 30 representa este componente.

Figura 30 – *Advanced Feeder.*



Depois de criados todos os componentes utilizados no Gêmeo Digital, iniciou-se a etapa de definição de *Behaviors* e *Properties*.

4.1.2 Definição de *Behaviors* e *Properties*

4.1.2.1 Implementação de *Properties*

Esta etapa iniciou com a implementação de certas *Properties* nos três componentes genéricos da peça de trabalho. Com isso, para cada um, foi definida uma variável global do

tipo *string* nomeada como "cor" e então seu valor foi editado para "*red*", "*metal*", "*black*". Dessa forma, estas variáveis poderiam ser utilizadas, posteriormente, pelo *Python Script*. Em sequência, os materiais de cada um desses componentes foram alterados para que visualmente correspondessem ao valor definido na variável "cor". As figuras 31, 32 e 33 ilustram os componentes das peças de trabalho e suas *Properties*.

Figura 31 – *Componente genérico da peça de trabalho vermelha.*

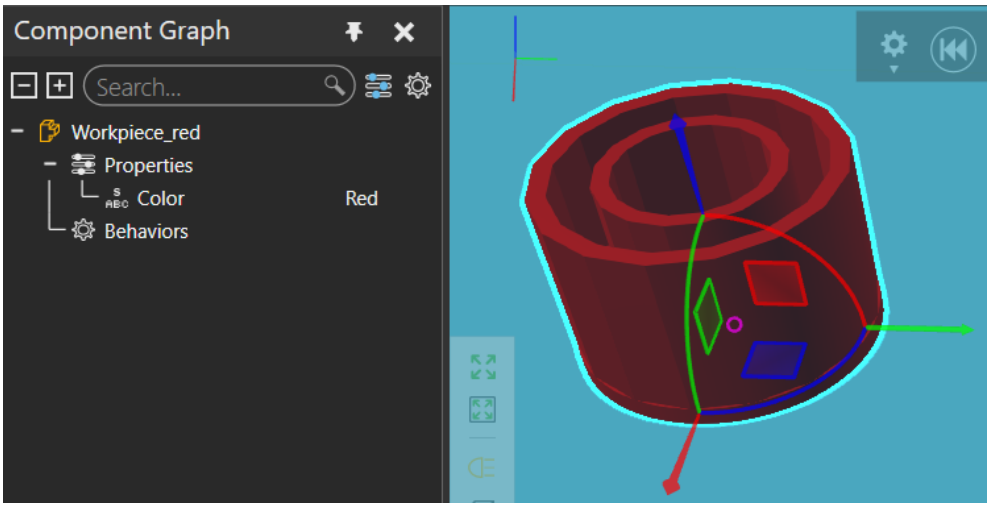
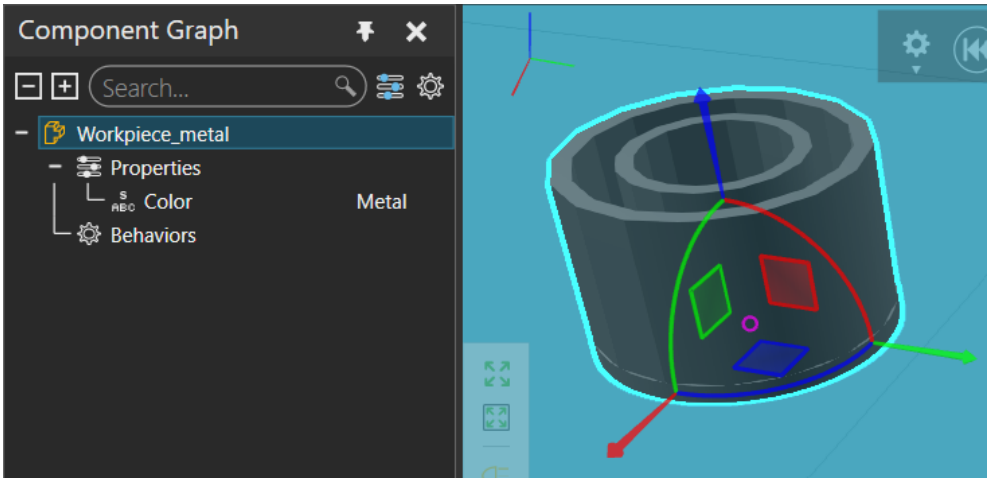
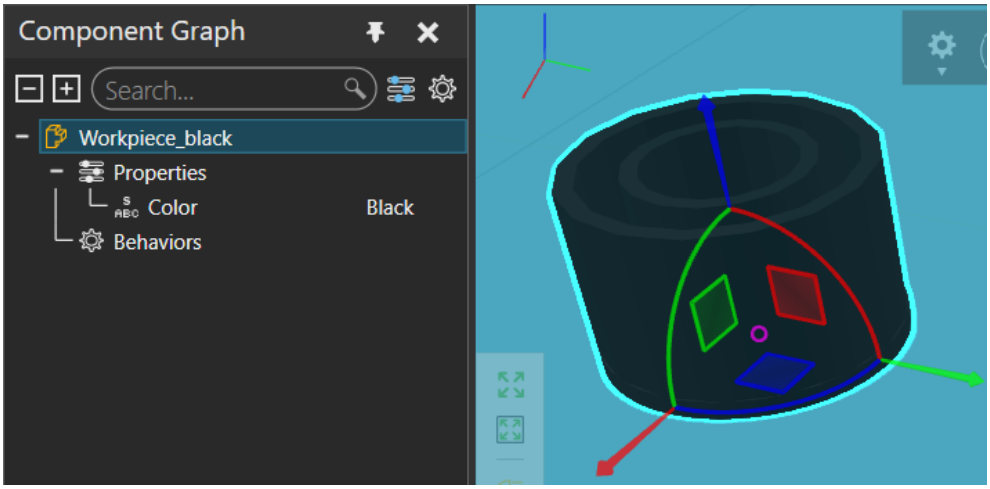


Figura 32 – *Componente genérico da peça de trabalho metálica.*



Após essas alterações, foi implementado uma *Property* no componente da esteira transportadora, sendo uma variável global do tipo *boolean* denominada "*Accumulate Mode*". A ideia por trás dessa variável é permitir que o usuário configure seu valor, deste modo, posteriormente o *Python Script* leria está variável e de acordo com seu valor, ativaria ou desativaria o modo de acúmulo de peças de trabalho nas rampas separadoras durante a simulação.

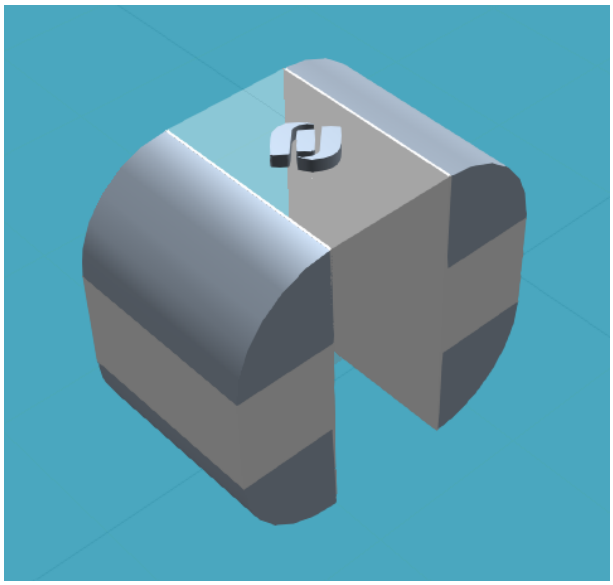
Figura 33 – *Componente da peça de trabalho preta.*



Sobre os componentes atuadores, suas restrições foram configuradas de acordo com o descrito no projeto. Para o módulo atuador de parada a propriedade de translação da junta prismática foi configurada com um deslocamento mínimo de 0 mm e máximo de 13 mm, sua velocidade e aceleração permaneceu a padrão do *software*. Para os módulos desviadores a propriedade de rotação da junta rotacional foi configurada com um deslocamento de no mínimo 0 graus e no máximo 55 graus, suas velocidades e acelerações também permaneceram a padrão.

Ademais, as propriedades de dimensão do componente *Advanced Feeder* foram alteradas para que visualmente se adequasse ao tamanho da estação. A Figura 34, demonstra como ficou este componente.

Figura 34 – *Advanced Feeder modificado.*



Finalizada a implementação e alteração de certas *Properties*, partiu-se para a imple-

mentação de *Behaviors*, iniciando com a definição de interfaces.

4.1.2.2 Definição de interfaces

Todas as interfaces adicionadas nesta etapa foram do tipo "*One To One*", isto é, se conectam apenas com uma outra interface. Primeiramente foram definidas as interfaces do componente central, a esteira transportadora. Iniciou-se a implementação com as interfaces de conexão física entre componentes, as quais determinam o sentido do fluxo da peça de trabalho.

Uma interface foi nomeada como "*BeginInterface*", foi configurada com sentido de entrada de material e definida sobre o *Frame* de nome "*BeginFrame*", no início da esteira transportadora. Esta interface seria conectada com a interface de saída de material do componente *Advanced Feeder*, possibilitando a passagem de peça de trabalho para a esteira transportadora.

Foram adicionadas mais três interfaces nomeadas respectivamente de "*EndInterface1*", "*EndInterface2*" e "*EndInterface3*", com a configuração de sentido de saída de material, definidas, respectivamente, pelos *Frames* "*ExitForRamp1Frame*", "*ExitForRamp2Frame*" e "*ExitForRamp3Frame*". Essas interfaces seriam conectadas com interfaces de entrada de material de cada rampa separadora. Possibilitando a saída das peças de trabalho para as rampas separadoras.

Depois da definição das interfaces de fluxo de material, foram implementadas as interfaces de sinais na esteira transportadora. Para estas, não foi necessário vinculá-las a um *Frame* do componente. Então, foram adicionadas três interfaces do tipo sinal, nomeadas de "*DI3_Aux1interface*", "*DI3_Aux2interface*" e "*DI3_Aux3interface*". Estas interfaces seriam vinculadas com três sinais os quais seriam criados e depois conectadas com outras interfaces de cada rampa separadora. Isso possibilita que algum sinal de uma rampa separadora que esteja vinculada a sua respectiva interface de sinal, esteja relacionado ao sinal da esteira transportadora que esteja vinculada a sua respectiva interface de sinal. Portando se um valor do sinal da rampa separadora muda, o sinal relacionado da esteira transportadora também muda.

Com as interfaces da esteira transportadora definidas. Partiu-se para a definição das interfaces das rampas separadoras.

As interfaces das rampas separadoras foram definidas de modo simétrico para cada rampa. Primeiramente foram implementadas as interfaces de fluxo de material e em sequência as interfaces de sinais.

Para cada rampa, definiu-se uma interface de fluxo de material com sentido de entrada, com os nomes respectivos de "*BeginRamp1Interface*", "*BeginRamp2Interface*" e "*Be-*

ginRamp3Interface", as quais foram vinculadas, respectivamente, aos *Frames* iniciais de cada rampa de nomes "*InitialRamp1Frame*", "*InitialRamp2Frame*" e "*InitialRamp3Frame*". Como comentado anteriormente, estas interfaces foram conectadas com as três interfaces respectivas de saída de material da esteira transportadora.

Ademais, para cada rampa, definiu-se uma interface de fluxo de material com sentido de saída para cada rampa com os nomes respectivos de "*EndRamp1Interface*", "*EndRamp2Interface*" e "*EndRamp3Interface*", as quais foram vinculadas, respectivamente, aos Frames finais de cada campa de nomes "*EndRamp1Frame*", "*EndRamp2Frame*" e "*EndRamp3Frame*". Estas interfaces seriam conectadas ou desconectadas com as interfaces de entrada de material dos componentes auxiliares comentados anteriormente, possibilitando assim a ativação ou desativação do modo de acúmulo de peças de trabalho nas rampas separadoras.

Prosseguiu-se, então, com a adição de uma interface de sinal para cada rampa, estas foram nomeadas de "*SinalAux1Interface*", "*SinalAux2Interface*" e "*SinalAux3Interface*". Assim como descrito anteriormente, essas interfaces foram conectadas com as três interfaces de sinais da esteira transportadora.

Com todas as interfaces das rampas separadoras definidas. Partiu-se para a definição das interfaces dos três componentes auxiliares. Para cada um destes componentes, foi adicionado apenas uma interface de fluxo de material com sentido de entrada. Os nomes dessas interfaces foram deixadas como padrão sendo "*OneToOneInterface*" e foram vinculadas aos *Frames* respectivos de cada componente de nomes "*In_flowFrame*".

Finalizado a definição de todas as interfaces. Iniciou-se a definição das rotas de fluxo de material, chamados de *paths*.

4.1.2.3 Definição dos *paths*

Todas as rotas adicionadas nesta etapa foram do tipo "*One Way Path*", isto é, existe apenas um sentido no fluxo de material. Em um primeiro momento foram definidos os *paths* do componente central, a esteira transportadora. Foram implementadas quatro dessas rotas neste componente. Elas seriam posteriormente declaradas no *Python Script*, onde executaria a lógica de qual caminho a peça de trabalho seguiria.

O primeiro *path* foi denominado como "*BasicPath*" e foi definido na respectiva ordem dos *Frames*: "*BeginFrame*" e "*ReadFrame*". Essa rota é acessada por todas as peças de trabalho que são produzidas pelo *Advanced Feeder* e entram na esteira transportadora.

Os outros três *paths*, foram denominados como "*Path1*", "*Path2*" e "*Path3*" e foram definidos na respectiva ordem dos *frames*: "*GenericInitialRampFrame*", "*InitalRamp1Frame*" e "*ExitForRamp1Frame*"; "*GenericInitialRampFrame*", "*InitalRamp2Frame*" e "*ExitForRamp2Frame*"; "*GenericInitialRampFrame*", "*InitalRamp3Frame*" e "*ExitForRamp3Frame*". A peça de trabalho é desiguinada para um desses caminhos pelo *Python Script* de acordo com a lógica de separação das cores introduzida por uma rede de Petri no emulador.

Concluído os *paths* da esteira transportadora. Implementou-se rotas para cada rampa separadora. Estas foram nomeadas respectivamente de "*Ramp1Path*", "*Ramp2Path*" e "*Ramp3Path*" e foram definidos pelos respectivos *Frames* de cada rampa: "*InitialRamp1Frame*", "*Ramp1SensorFrame*" e "*EndRamp1Frame*"; "*InitialRamp2Frame*", "*Ramp2SensorFrame*" e "*EndRamp2Frame*"; "*InitialRamp3Frame*", "*Ramp3SensorFrame*" e "*EndRamp3Frame*". As peças de trabalho que saíssem da esteira transportadora entrariam respectivamente nos *paths* da cada rampa separadora.

Tendo as rotas das rampas finalizadas, implementou-se as rotas dos três componentes auxiliares. As rotas desses componentes foram deixadas com o nome padrão de "*OneWayPath*" e foram definidas pelo *Frame* "*In_flowFrame*" respectivo de cada um desses componentes. Como comentado anteriormente, essas rotas foram configuradas para ter capacidade zero, permitindo assim aplicar a lógica de acúmulo de peças de trabalho nas rampas separadoras.

Finalizado a definição de todos os *paths*. Iniciou-se a definição dos sensores.

4.1.2.4 Definição de sensores

Assim como definido pelo projeto, foram adicionados cinco sensores do tipo *Raycast*. Dois destes sensores foram implementados no componente da esteira transportadora e os outros três foram implementados um em cada rampa separadora.

O primeiro sensor da esteira transportadora foi denominado como "*SensorOpticoDifuso1*" e definido no *Frame* "*BeginFrame*" na direção do primeiro sensor óptico difuso com o intuito de ser ativado se houver alguma peça de trabalho no início da esteira, de forma análoga ao sensor real. O segundo sensor foi denominado "*SensorConjuntoLeCores*" e definido no *Frame* "*ReadFrame*" na direção do segundo sensor óptico difuso e do sensor indutivo com o intuito de ser ativado quando a peça de trabalho chegar na posição de leitura dos sensores.

Com relação aos três sensores aplicados nas rampas separadoras, estes foram denominados como "*SensorRetroreflexivoParte1*", "*SensorRetroreflexivoParte2*" e "*SensorRetroreflexivoParte3*", foram definidos nos *Frames* respectivos de cada rampa: "*Ramp1SensorFrame*",

"*Ramp2SensorFrame*" e "*Ramp3SensorFrame*", na mesma direção do sensor retro-reflexivo. Dessa forma, quando alguma peça cruzar qualquer um destes sensores seria como se o sensor retro-reflexivo real fosse ativado.

Concluído a definição de todos os sensores. Iniciou-se a definição dos sinais.

4.1.2.5 Definição de sinais

Como definido pelo projeto, a implementação de sinais dividiu-se em dois grupos: os sinais de entrada e saída, respectivos aos terminais *I/Os* da tabela 1, e os sinais auxiliares para funcionamento da simulação. Todos os sinais definidos foram vinculados com o *Python Script* e alguns destes também foram vinculados com os sensores definidos anteriormente, isto significa que se o sensor for acionado pela passagem de uma peça de trabalho, o sinal vinculado a esse sensor é ativado, caso o sensor não seja acionado o sinal permanece desativado.

Iniciou-se com a definição do primeiro grupo de sinais. Todos estes sinais são do tipo *boolean* e foram adicionados ao componente central, esteira transportadora. Logo, os sinais de saída da simulação e entrada para o emulador foram nomeados como: "*DI0_Peça disponível no inicio da esteira*", "*DI1_Peça metálica*", "*DI2_Peça não preta*", "*DI3_Rampa cheia*", "*DI4_Atuador 1 recuado*", "*DI5_Atuador 1 avançado*", "*DI6_Atuador 2 recuado*" e "*DI7_Atuador 2 avançado*". Os sinais de saída do emulador e entrada para a simulação foram nomeados como: "*DO0_Liga esteira*", "*DO1_Avança atuador 1*", "*DO2_Avança atuador 2*", "*DO3_Recua atuador de parada*" e "*DO7_Estação ocupada*". Ademais, o sinal denominado "*DI0_Peça disponível no inicio da esteira*" foi vinculado com sensor "*SensorOpticoDifuso1*" implementado anteriormente.

O segundo grupo de sinais foi definido em sequência. Para o componente da esteira transportadora foram criados quatro sinais auxiliares do tipo *boolean* e um sinal do tipo *component*. O primeiro sinal auxiliar do tipo *boolean* foi denominado "*Sinal_AuxBoolean*" e o sinal do tipo *component* foi denominado "*Sinal_Componente*", ambos foram vinculados ao sensor "*SensorConjuntoLeCores*". Quando este sensor for acionado, além de ativar o sinal "*Sinal_AuxBoolean*" ele ativará o sinal de *component* "*Sinal_Componente*" que passará a ter em seu valor o componente que acionou o sensor, esta informação será usada no *Python Script* para possibilitar a leitura das propriedades de cor, definidas anteriormente, das peças de trabalho. Os outros três sinais *boolean* auxiliares foram denominados como "*DI3_Aux1*", "*DI3_Aux2*" e "*DI3_Aux3*", estes foram vinculados respectivamente as interfaces de sinais "*DI3_Aux1interface*", "*DI3_Aux2interface*" e "*DI3_Aux3interface*", definidas anteriormente.

Com os sinais auxiliares da esteira transportadora adicionados. Foram implementados um sinal auxiliar do tipo *boolean* para cada um dos três componentes de rampa transportadora. Estes foram denominados respectivamente como "*SinalAux1*", "*SinalAux2*" e "*SinalAux3*". Estes sinais foram vinculados, respectivamente, às interfaces de sinais de nomes "*SinalAux1Interface*", "*SinalAux2Interface*" e "*SinalAux3Interface*" e também aos sensores de de nomes "*SensorRetro-reflexivoParte1*", "*SensorRetroreflexivoParte2*" e "*SensorRetroreflexivoParte3*" de cada rampa. Portanto, caso alguma peça acione algum destes sensores, o sinal auxiliar da rampa correspondente seria ativado, como este sinal está vinculado a interface de sinal da rampa e esta por sua vez está conectada com a interface de sinal respectiva da esteira transportadora, o sinal auxiliar correspondente da esteira transportadora vinculada a sua interface também seria ativada. Logo, os sinais auxiliares "*DI3_Aux1*", "*DI3_Aux2*" e "*DI3_Aux3*" da esteira transportadora assumem, respectivamente, o mesmo valor dos sinais auxiliares de cada rampa "*SinalAux1*", "*SinalAux2*" e "*SinalAux3*". Estes sinais auxiliares foram relacionados com o sinal "*DI3_Rampa cheia*" por meio do *Python Script*, desta forma, se qualquer um dos sinais auxiliares das rampas forem ativados, o sinal final "*DI3_Rampa cheia*" também será ativado, simulando o acionamento do sensor retro-reflexivo.

Finalizada a definição de todos os sinais. Iniciou-se a definição dos servo controladores.

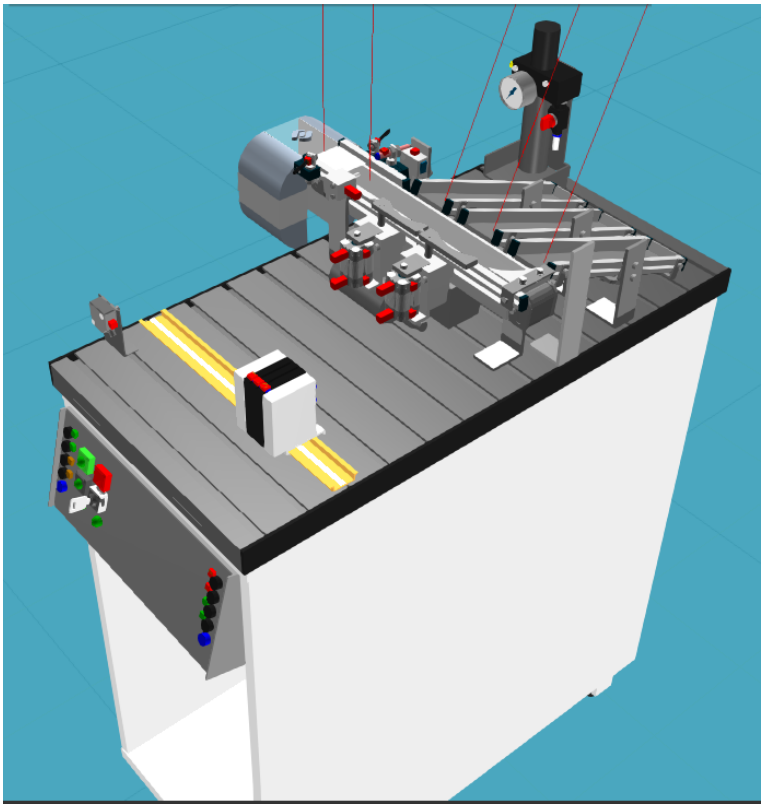
4.1.2.6 Definição de servo controladores

Assim como definido pelo projeto, foram adicionados *Behaviors* do tipo servo controlador a todos os componentes atuadores, estes são: atuador linear e os dois módulos desviadores. Esses servo controladores foram configurados para as juntas de cada componente atuador. A adição destes *Behaviors* permite executar uma função no *Python Script* que movimenta as juntas na simulação.

Concluída a definição dos servo controladores, a segunda etapa da implementação foi finalizada. Ademais adicionou-se um *Behavior* de *Python Script* ao componente central, esteira transportadora, o qual foi editado na etapa seguinte da implementação. Com isso a estação final é ilustrada na Figura 35. As linhas verticais vermelhas que aparecem na figura, indicam os cinco sensores *Raycast*, definidos anteriormente. Os três componentes auxiliares já não aparecem mais na estação por terem sido configurados como não visíveis.

O *Component Graph* dos dez componentes manipulados depois da implementação de todos os *Behavior* e *Properties* pode ser observado nas figuras 36, 37, 38 e 39.

Figura 35 – *Configuração final da bancada.*



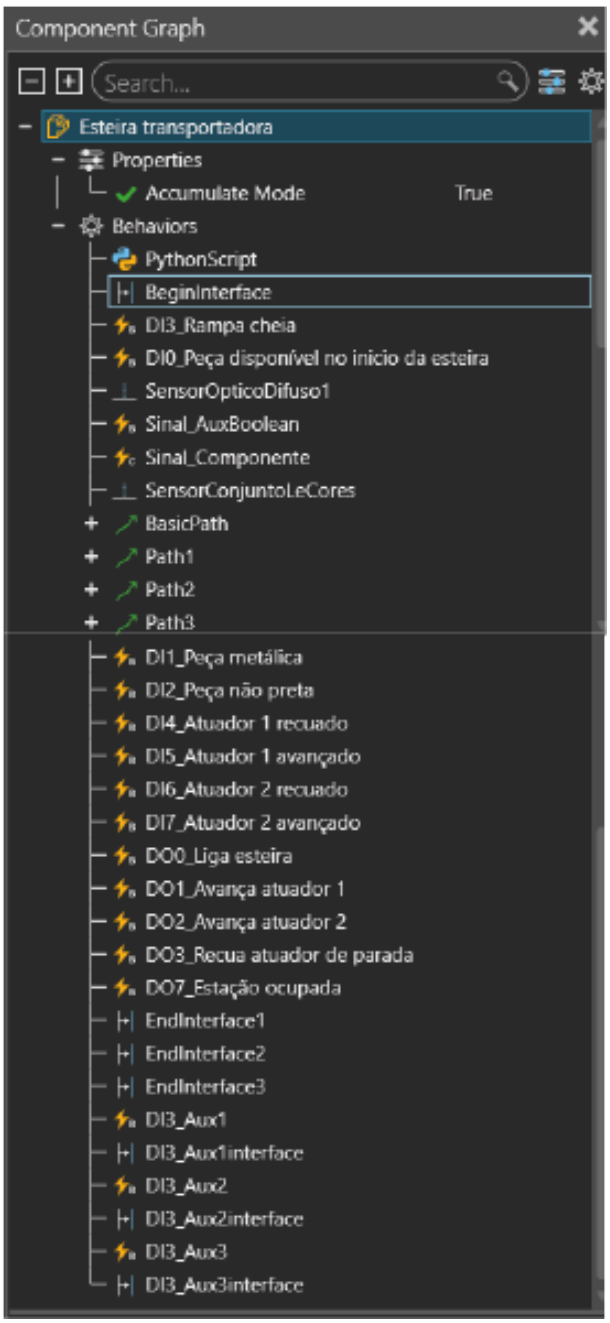
4.1.3 Implementação do algoritmo

Nesta terceira etapa iniciou-se a edição do *Python Script* adicionado anteriormente na esteira transportadora. Como descrito no projeto, a função deste *Behavior* é implementar a lógica de mudanças e as funcionalidades dos sinal de entrada e saída. Os diferentes componentes, seus *Behavior* e *Properties* foram declarados no algoritmo para que pudessem ser manipulados para o funcionamento correto do Gêmeo Digital.

A estrutura inicial do *Python Script* já veio com duas funções pré-definidas, a primeira função denominada "*OnSignal*", aciona quando algum *Behavior* de sinal conectado ao *script* muda seu valor, e a segunda função denominada "*OnRun*", refere-se ao *loop* principal do código, em que a simulação é estabelecida e funcionalidades de movimentação dos atuadores são aplicadas. A movimentação dos atuadores é executada pela função "*moveJoint*", inerente a objetos do tipo servo controladores. Ademais, foram declarados no algoritmo, os três servo controladores dos componentes atuadores. No caso deste algoritmo, os movimentos aplicados para os atuadores limitam-se em duas possibilidades, a extensão máxima ou mínima da junta, estabelecida pelas restrições impostas anteriormente.

A lógica implementada no algoritmo para as funcionalidades da estação sobre os 18

Figura 36 – *Component Graph da esteira transportadora.*



sinais definidos na esteira transportadora é explicada a seguir. Primeiramente, aborda-se sobre os sinais de saída da simulação e os auxiliares relacionados. Em sequência, aborda-se sobre os sinais de entrada para a simulação.

O primeiro sinal "*DI0_Peça disponível no inicio da esteira*" não possui lógica direta no código, pois a mudança deste já é causada pela passagem de peça pelo *Behavior* de sensor "*SensorOpticoDifuso1*" definido anteriormente.

O sinal "*DI1_Peça metálica*" e "*DI2_Peça não preta*" estão relacionados com os sinais auxiliares "*Sinal_AuxBoolean*" e "*Sinal_Componente*". Estes dois sinais auxiliares estão vincu-

Figura 37 – *Component Graph dos componentes atuadores.*

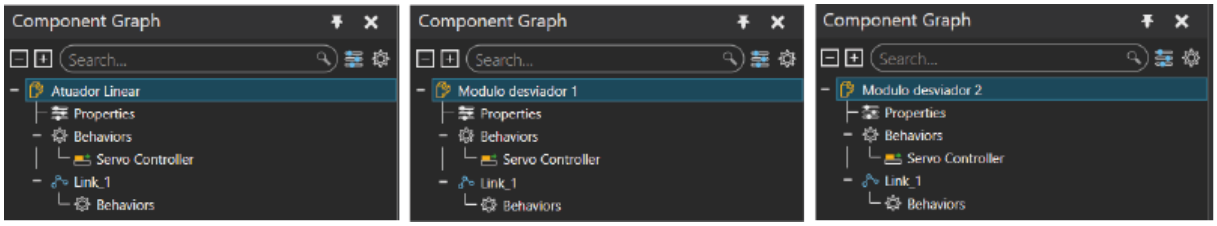


Figura 38 – *Component Graph dos componentes de rampa transportadora.*

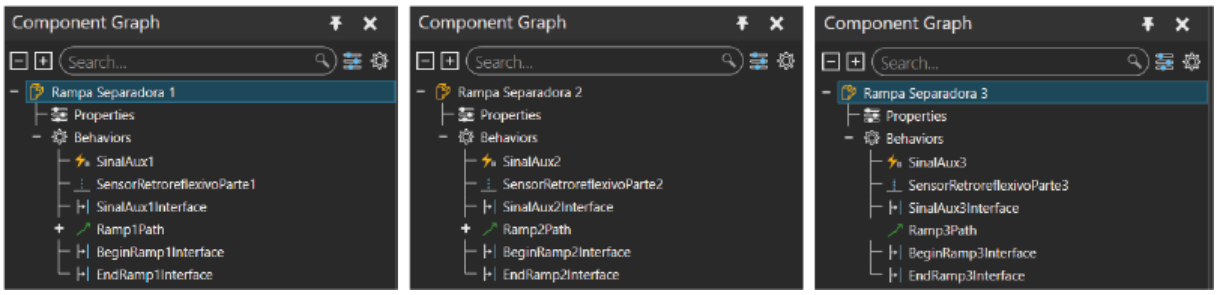


Figura 39 – *Component Graph dos componentes auxiliares.*

lados ao sensor "*SensorConjuntoLeCores*", quando este é acionado o valor do sinal auxiliar "*Sinal_AuxBoolean*" é ativado. Implementou-se um condicional deste sinal na função "*OnSignal*", de modo que o *script* lê o valor do sinal do tipo *component* "*Sinal_Componente*" no momento em que o sinal "*Sinal_AuxBoolean*"for ativado. Este valor lido consiste na peça de trabalho que acionou o sensor. O *script* acessa o valor da propriedade cor definida nas peças de trabalho, caso seja "*red*", os sinais "*DI1_Peça metálica*" e "*DI2_Peça não preta*" são configurados para "*False*" e "*True*", respectivamente. Caso seja "*metal*" os sinais "*DI1_Peça metálica*" e "*DI2_Peça não preta*" são configurados para "*True*" e "*True*", respectivamente. Por fim, caso seja "*black*" os sinais "*DI1_Peça metálica*" e "*DI2_Peça não preta*" são configurados para "*False*" e "*False*", respectivamente.

Ademais, para os sinais auxiliares "*DI3_Aux1*", "*DI3_Aux2*" e "*DI3_Aux3*" implementou-se um condicional para cada um na função "*OnSignal*". Assim como explicado anteriormente, quando algum dos sinais auxiliares das rampas separadoras forem alterados, o respectivo sinal desses auxiliares também será modificado, passando pela função "*OnSignal*". O condicional

definido determina que se qualquer um dos sinais "*DI3_Aux1*", "*DI3_Aux2*" e "*DI3_Aux3*", forem ativados o sinal "*DI3_Rampa cheia*" recebe "*True*", caso forem desativados o sinal "*DI3_Rampa cheia*" recebe "*False*".

Prosseguindo para os sinais "*DI4_Atuador 1 recuado*" e "*DI5_Atuador 1 avançado*". Estes sinais são configurados na função "*OnRun*". Quando a função "*moveJoint*" é executada para avançar o módulo desviador 1, antes de iniciar o movimento na simulação, o sinal "*DI4_Atuador 1 recuado*" é configurado para "*False*", após a execução do movimento o sinal "*DI5_Atuador 1 avançado*" é configurado para "*True*". Caso a função "*moveJoint*" é executada para recuar o módulo desviador 1, antes de iniciar o movimento na simulação, o sinal "*DI5_Atuador 1 avançado*" é configurado para "*False*", após a execução do movimento o sinal "*DI4_Atuador 1 recuado*" é configurado para "*True*". Isso ocorre de forma análoga aos sinais "*DI6_Atuador 2 recuado*" e "*DI7_Atuador 2 avançado*" para o módulo desviador 2.

Agora, discorre-se sobre os sinais de entrada para a simulação. O primeiro sinal "*DO0_Liga esteira*" é contemplado com um condicional na função "*OnSignal*", quando o emulador alterar o sinal da variável análoga a esta, a condição será satisfeita. Caso o valor deste sinal for para "*False*", o *script* desabilita todos os *paths* "*BasicPath*", "*Path1*", "*Path2*" e "*Path3*", fazendo com o que qualquer peça que estiver em alguma dessas rotas pare de se mover. Caso o valor mude para "*True*", o *script* habilita estes *paths*, fazendo que as peças nestas rotas comecem a se mover.

Com relação ao sinal "*DO1_Avança atuador 1*" implementou-se um condicional na função "*OnRun*", que analisa este sinal. Caso ele seja "*False*", uma função "*moveJoint*" será executada para mover a junta do módulo desviador 1 para a posição mínima, caso o sinal for "*True*" uma função "*moveJoint*" será executada para mover sua junta para a posição máxima. Isso ocorre de forma análoga ao sinal "*DO2_Avança atuador 2*" só que para o módulo desviador 2.

Sobre o sinal "*DO3_Recua atuador de parada*", também implementou-se um condicional na função "*OnRun*". Caso o sinal seja "*False*", uma função "*moveJoint*" será executada para mover a junta do módulo atuador de parada para a posição máxima, caso seja "*True*", uma função "*moveJoint*" será executada para mover sua junta para a posição mínima, ou seja, junta recuada.

Por fim, o sinal "*DO7_Estação ocupada*" foi contemplada com um condicional na função "*OnSignal*". Caso seu valor mude para "*False*", o material do componente *Led* será alterado para verde, e o material de uma seção do componente *Advanced Feeder* também será

modificado para verde, indicando que a estação está livre para a entrada de peças. Além do mais o *path* do *Advanced Feeder* é habilitado, permitindo a passagem da peça de trabalho para a esteira transportadora. Caso o valor do sinal mude para "*True*", o material do *Led* será alterado para vermelho e o material da seção do *Advanced Feeder* também, indicando que a estação está ocupada. Ademais, o *path* do *Advanced Feeder* é desabilitado, impedindo a passagem da peça de trabalho para a esteira transportadora.

Com isso, a lógica de todos os sinais foi contemplada pelo *Python Script*. Então, aborda-se sobre a lógica de roteamento das peças de trabalho. Essa se dá de modo condicional na função "*OnRun*". Caso o sinal "*DO1_Avança atuador 1*" esteja "*True*" a peça de trabalho segue pela rota "*Path1*", entrando para a rampa separadora 1. Caso o sinal "*DO2_Avança atuador 2*" esteja "*True*" e o sinal "*DO1_Avança atuador 1*" esteja "*False*", a peça de trabalho segue pela rota "*Path2*" entrando para a rampa separadora 2. Caso estes sinais estejam "*False*", a peça segue pela rota "*Path3*", entrando para a rampa separadora 3.

Por fim, foi definida a lógica de funcionamento do modo acumulativo comentado anteriormente. Na função "*OnRun*", antes de iniciar o *loop* da simulação, é feita uma tratativa que verifica o valor da *Property* "*Accumulate Mode*" definida anteriormente. Este valor pode ser alterado pelo usuário antes de iniciar a simulação. Caso o valor seja "*True*", o *script* conecta as interfaces de saída de fluxo da peça de trabalho das rampas separadoras, com as respectivas interfaces de entrada de fluxo de peça de trabalho dos componentes auxiliares, como a capacidade dos *paths* dos componentes auxiliares é zero, as peças de trabalho começam a acumular nas rampas separadoras. Caso o valor dessa *Property* seja "*False*", o *script* desconecta essas interfaces, fazendo com que as peças de trabalho que chegam no final das rampas separadoras desapareçam da simulação.

Isso conclui a explicação de modo geral do funcionamento do algoritmo implementado. Em sequência, aborda-se sobre a conectividade OPC-UA .

4.1.4 Conectividade OPC-UA

Esta última etapa da implementação do Gêmeo Digital corresponde à implantação da conexão entre o Gêmeo Digital e o emulador de rede de Petri. Esta conexão foi estabelecida pelo protocolo de comunicação OPC-UA, por meio do API OPC-UA disponível no *Visual Components*. Para isso foi necessário primeiramente entrar com o endereço do servidor do emulador, estabelecendo a conexão, para que então fosse possível parear os sinais do Gêmeo

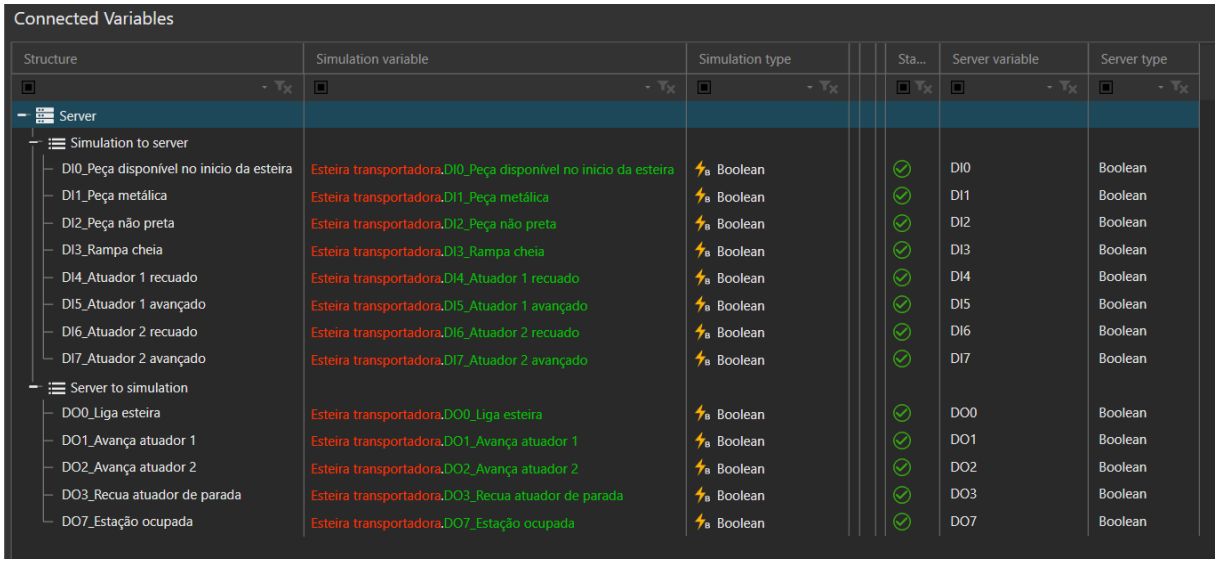
Digital com os do emulador.

Primeiramente os sinais de saída do Gêmeo Digital e entrada do emulador foram pareados. Os pares de sinais ficaram "*DI0_Peça disponível no inicio da esteira*" com "*DI0*", "*DI1_Peça metálica*" com "*DI1*", "*DI2_Peça não preta*" com "*DI2*", "*DI3_Rampa cheia*" com "*DI3*", "*DI4_Atuador 1 recuado*" com "*DI4*", "*DI5_Atuador 1 avançado*" com "*DI5*", "*DI6_Atuador 2 recuado*" com "*DI6*" e "*DI7_Atuador 2 avançado*" com "*DI7*".

Depois, seguiu-se com o pareamento dos sinais de entrada do Gêmeo Digital e saída do emulador. Os pares de sinais ficaram "*DO0_Liga esteira*" com "*DO0*", "*DO1_Avança atuador 1*" com "*DO1*", "*DO2_Avança atuador 2*" com "*DO2*", "*DO3_Recua atuador de parada*" com "*DO3*" e "*DO7_Estação ocupada*" com "*DO7*".

A Figura 40 mostra o painel de conexão OPC-UA do *Visual components* com as variáveis do Gêmeo Digital e o emulador pareadas.

Figura 40 – *Painel de conexão OPC-UA.*



Fonte: (Visual Components, 2021)

4.2 Emulador de rede de Petri

Para a construção do Emulador de rede de Petri, a fim de funcionar conforme proposto neste trabalho, foi segmentado o código em três frentes, executadas de forma praticamente concorrente tal qual é feita com a biblioteca Trio (SMITH. N. J., 2017) de Python. As três frentes são: Interface do Usuário (responsável por receber informações do usuário, bem como apresentar informações relevantes), Interpretador de Arquivo XML e rede de Petri (responsável

por alterar os estados de forma coerente ao arquivo inserido pelo usuário) e o servidor OPC-UA (responsável por estabelecer a conexão correta com o Gêmeo Digital). Para que as três frentes funcionassem a partir das mesmas informações, foi criada uma Classe em Python denominada *ExchangeableVariables*, que armazena todas as informações atuais do programa em um único endereço na memória e é passada como parâmetro das três funções citadas a cima, permitindo alterações específicas em cada uma delas, mantendo a coerência da informação entre elas.

4.2.1 Interface do Usuário

A Interface do Usuário (Figura 41) foi desenvolvida com o auxílio da biblioteca PySimpleGui (PySimpleGUI, 2020) e é separada em 4 seções.

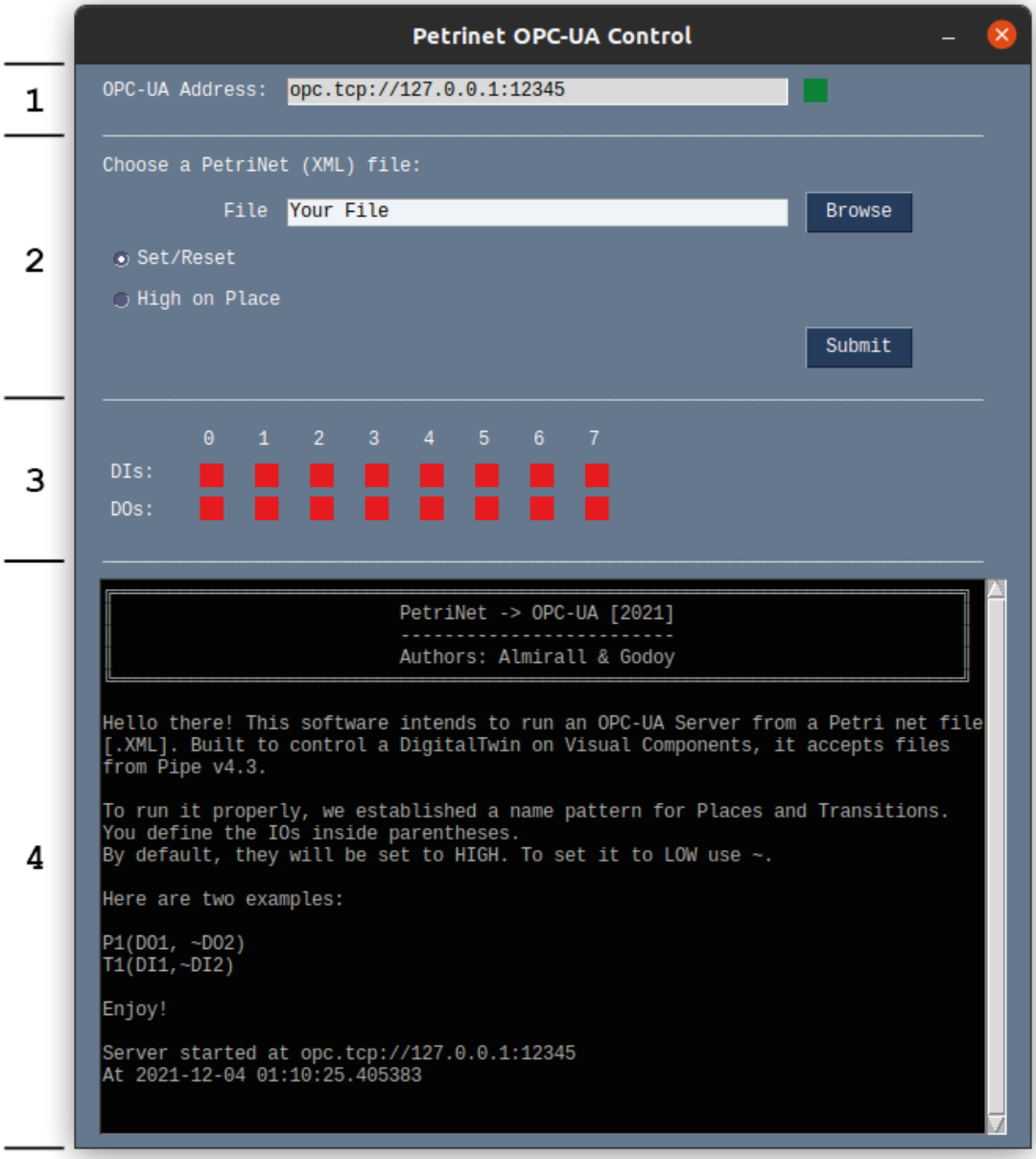
A primeira diz respeito ao endereço em que o Servidor OPC-UA será estabelecido, seguido de um quadrado que se mantém vermelho enquanto o Servidor estiver inativo, e verde enquanto o Servidor estiver ativo.

A segunda dá opção ao usuário escolher um arquivo, necessariamente no formato XML. Ao clicar o botão "*Browse*", abre-se uma janela que permite ao usuário navegar pelos seus diretórios até encontrar o arquivo desejado. Há ainda, nesta seção, duas opções para a forma como a rede de Petri é interpretada. A opção "*Set/Reset*" faz com que as saídas sejam alterados exatamente conforme é definido nos Lugares da rede de Petri, sendo necessário, portanto, que o usuário informe quando uma saída é redefinida para zero. Já a opção "*High on Place*" faz com que apenas as saídas estabelecidas nos Lugares com Marcas sejam definidas para um, ou seja, assim que a Marca é consumida e o Lugar fica vazio a saída é alterada para zero, dando flexibilidade para o usuário escolher como modelar a rede de Petri. Escolhido o arquivo e o modo de execução, o usuário deve apertar o botão "*Submit*" para que o arquivo seja lido.

A terceira tem papel informativo. Mostra o atual estado de cada uma das Entradas e Saídas do Servidor. Quadrados vermelhos representam o sinal em zero (*Low*), e quadrados verdes representam o sinal em um (*High*).

A quarta diz respeito aos registros da execução. Inicia com uma breve introdução ao programa, informa a inicialização do Servidor bem como o carregamento do arquivo XML e apresenta a ativação das transições, bem como pausas causadas por temporizadores nas Transições da rede de Petri.

Figura 41 – *Interface do Usuário.*



4.2.2 Interpretador de Arquivo XML e rede de Petri

A construção do Emulador de rede de Petri parte da leitura de um arquivo XML construído a partir do *software* PIPE (Platform Independent Petri net Editor, 2009). Utilizou-se um pacote nativo de Python chamado "xml" para fazer a leitura do arquivo e processar os diferentes atributos classificados com marcas como: <*place*>, <*transition*> e <*arc*>.

Para a manipulação e armazenamento da rede de Petri de forma mais organizada, aproveita-se da característica da linguagem Python em ser Orientada a Objetos para a criação de Classes específicas para cada parte essencial da rede de Petri.

Implementou-se, portanto, três Classes chamadas "*Place*", "*Transition*", "*Arc*", a fim de

armazenar as informações essenciais de cada tipo, dando sentido a rede de Petri. Vale ressaltar que a Classe "*Arc*" tem nos atributos "*Arc.source*" e "*Arc.target*", exatamente o endereço conectando Objetos do tipo Lugar a Transições e vice-versa, o que agiliza o acesso dos mesmos para atualização de parâmetros na execução da rede.

Implementou-se também uma classe responsável por, a partir do endereço do arquivo XML a ser interpretado, realizar a leitura, atribuindo aos objetos com seus respectivos tipos, as informações necessárias para criar um objeto contento todos os elementos da rede de Petri.

Essa estrutura, testada para diversos arquivos do *Software* PIPE, mostrou-se bastante eficiente para armazenar toda a informação contida na rede de Petri e, para uma rede de Petri como a exibida na Figura 13, apresenta *output* tal qual apresentado no Código 1.

Código 1 - *Execução teste de interpretação de Arquivos XML.*

```
In [1]: petri = Petrinet(
            'petrinet_examples/petrinet_example.xml'
        )
        petri.__dict__
Out[1]: {'places': [P0, P1], 'transitions': [T0], 'arcs': [P0→T0, T0→P1]}

In [2]: petri.arcs[0].__dict__
Out[2]: {'id': 'P0 to T0', 'source': P0, 'target': T0}

In [3]: petri.arcs[0].source.__dict__
Out[3]: {'id': 'P0', 'capacity': 1}
```

Para permitir a usabilidade do programa, foi definida uma regra para a nomenclatura dos Lugares e Transições, responsáveis pela correta interpretação e alteração das variáveis transacionadas via OPC-UA. Como uma rede de Petri convencional, para a ativação das transições ocorre somente se as marcas necessárias se encontram nos lugares corretos, anteriores à transição, bem como se são satisfeitas as condições da transição em si.

Sobre a convenção de nomenclatura, considerando a aplicação em questão, definiu-se que transições devem conter, entre parenteses as entradas necessárias para sua ativação. Entradas que começam com o carácter til ($\sim$) requerem o sinal em "0", enquanto entradas que não começam com tal carácter, requerem o sinal em "1". Em caso de mais de uma entrada, o

usuário deve separá-las com vírgula (,).

Assim, é possível definir os requisitos para ativação da transição, conforme Tabela 2.

Tabela 2 – Entradas Digitais.

Entrada	Requisitado como 1	Requisitado como 0
DI0	DI0	∼DI0
DI1	DI1	∼DI1
DI2	DI2	∼DI2
DI3	DI3	∼DI3
DI4	DI4	∼DI4
DI5	DI5	∼DI5
DI6	DI6	∼DI6
DI7	DI7	∼DI7

Alguns dos possíveis nomes de Transição, conforme explicado na Tabela 2 podem ser conferidos na Tabela 3.

Tabela 3 – Exemplos de Entradas Digitais.

Nome da transição	Requisito para ativação
T1(DI0)	DI0 = 1
T2(∼DI1)	DI1 = 0
T3(DI2, DI3)	DI2 = 1 E DI3 = 1
T4(DI4, ∼DI5)	DI4 = 1 E DI5 = 0

As transições ainda podem ser temporizadas. Para isso é necessário alterar, no PIPE, a informação denominada "*Timer*", definindo o tempo a ser esperado na seção "*Rate*" das opções da transição. Quando acionada, uma transição deste tipo coloca em espera a rede de Petri, mantendo possível ainda a interação do usuário com a interface, executando o consumo das Marcas e inserindo novas Marcas nos Lugares de destino assim que acabada a espera.

Seguindo o mesmo racional, os nomes dos Lugares devem conter, também entre parenteses, os nomes das saídas e seus respectivos sinais a serem definidos, conforme o mesmo padrão de uso do caractere til (∼). (Tabela 4)

Alguns dos possíveis nomes de Lugares, conforme explicado na Tabela 4 podem ser conferidos na Tabela 5.

Criada a rede de Petri, passa-se a parte de controlar as mudanças de estados a partir do recebimento de dados do sistema, neste caso, via OPC-UA.

Dada a informação de uma transição, foi implementada uma função responsável por elencar os potênciais arcos a serem ativados. Como os arcos na rede de Petri são unidirecionais, implementou-se uma função que avalia somente os atributos "*Arc.source*".

Tabela 4 – Saídas Digitais.

Saída	Definida como 1	Definida como 0
DO0	DO0	∼DO0
DO1	DO1	∼DO1
DO2	DO2	∼DO2
DO3	DO3	∼DO3
DO4	DO4	∼DO4
DO5	DO5	∼DO5
DO6	DO6	∼DO6
DO7	DO7	∼DO7

Tabela 5 – Exemplos de Saídas Digitais.

Nome do lugar	Define as seguintes saídas
P1(DO0)	DO0 = 1
P2(∼DO1)	DO1 = 0
P3(DO2, DO3)	DO2 = 1 E DO3 = 1
P4(DO4, ∼DO5)	DO4 = 1 E DO5 = 0

Deste modo, o emulador constantemente avalia os sinais nos *I/Os*, definindo as Transições passíveis de ativação. Assim que uma ou mais transições têm suas condições satisfeitas, o emulador ativa uma das Transições, o que altera os sinais nos *I/Os* e faz com o que emulador novamente avalie todas as transições da rede de Petri, repetindo o ciclo.

4.2.3 Servidor OPC-UA

Como parte dos requisitos do projeto, foi estabelecido um Servidor OPC-UA capaz de se conectar ao Gêmeo Digital. Para isso, utilizou-se da biblioteca de Python *freeOPCUA* (freeOPCUA, 2020), cuja classe Server permite a implementação de um Servidor OPC-UA na rede local do computador. Para tal, definiu-se o endereço padrão: "opc.tcp://127.0.0.1:12345". Além disso, na inicialização do Servidor, definiu-se todas as entradas digitais, bem como todas as saídas digitais, respeitando o padrão acordado com o Gêmeo Digital (DI0, DI1, DI2, DI3, DI4, DI5, DI6, DI7, DO0, DO1, DO2, DO3, DO4, DO5, DO6, DO7). Esta parte do programa é responsável por alterar os sinais de acordo com as informações do parâmetro "*VariableExchange*", sobrescrevendo os valores de Saídas Digitais a cada iteração. É responsabilidade do Gêmeo Digital, por sua vez, alterar os valores das Entradas Digitais, que são lidos e atualizam o "*VariableExchange*", também a cada iteração.

4.2.4 Armazenamento de Variáveis

Como descrito na introdução desta seção, para que ocorra a troca de informações de forma correta entre a interface, o emulador de rede de Petri e o servidor OPC-UA; implementou-se uma Classe denominada "*VariableExchange*", que carrega os seguintes parâmetros, passíveis de alteração. (Tabela 6)

Tabela 6 – "*VariableExchange*"

Variável	Tipo	Descrição	Alterável por
DI	Dicionário	Chaves como entradas (ex.: DI1) e estado atual como valor.	OPC-UA
DO	Dicionário	Chaves como entradas (ex.: DO1) e estado atual como valor.	rede de Petri
address	String	opc.tcp://127.0.0.1:12345	OPC-UA
file_xml	String	/petrinet_example.xml	UI
set_reset	Booleana	Responsável por definir a forma como a rede de Petri será interpretada.	UI
running	Booleana	Responsável por definir se o programa está aberto e executando.	UI
connected	Booleana	Responsável por definir se o Servidor se conectou com sucesso.	OPC-UA
petrinet_ready	Booleana	Responsável por definir se o arquivo XML foi interpretado corretamente.	rede de Petri
gui_ready	Booleana	Responsável por definir se a Interface do Usuário foi utilizada corretamente.	UI
opcua_ready	Booleana	Responsável por definir se o Servidor se conectou com sucesso.	OPC-UA

4.2.5 Execução

Ao executar o programa, o usuário perceberá que a interface estabelecerá um Server OPC-UA no endereço informado. Dado que a definição das Entradas e Saídas já estará estabelecida no Gêmeo Digital, cabe ao usuário selecionar, no Visual Components, a conexão OPC-UA e a ativar. Inicializando a simulação, o programa começará a responder ao arquivo XML selecionado e, se feito corretamente, mostrará no Gêmeo Digital exatamente o que foi instruído pela rede de Petri. Podendo, inclusive, ser usado como validador da rede de Petri.

5 Testes e Resultados

Este capítulo documenta os testes realizados para garantir o funcionamento de ambas as frentes desenvolvidas. Busca mostrar teste individuais entre partes modulares mas também testes completos envolvendo todas as frentes do trabalho.

Implementadas as etapas do programa, foram testadas cada uma das frentes (Interface do Usuário, Interpretador de Arquivo rede de Petri em XML e Servidor OPC-UA) de forma separada, simplificando a interação entre as diferentes funções. Para a Interface do Usuário, verificou-se o valor recebido pelo evento de submissão de um arquivo.

Já para o Interpretador de Arquivo rede de Petri em XML, foram testadas redes de Petri bastante diversas a partir da instanciação repetidamente da classe *Petrinet*. Por se tratar de um arquivo XML padronizado na versão do PIPE definida, verificou-se consistência entre todas as redes de Petri inseridas e modeladas conforme as classes descritas.

Assim, foi validada esta etapa do programa. Para o Servidor OPC-UA, criou-se, da mesma maneira, um servidor enxuto contento 8 entradas digitais e 8 saídas digitais. Feito isso, buscou-se estabelecer conexão com o programa de testes de Servidor OPC-UA denominado UA-Expert (UA-Expert, 2018), que exibiu tanto o Servidor em si, na ocasião denominado "OPCUA_DIGITAL_TWIN", quanto as 8 entradas digitais e 8 saídas digitais, o que pode ser conferido na Figura 42, sendo um sucesso quando acessado via rede local. Sendo ainda possível testar a alteração de variáveis no Servidor, sendo refletidas no Cliente ou ainda alterações nas variáveis no Cliente sendo refletidas no Servidor, tal qual é esperado da interação entre o interpretador e o Gêmeo Digital.

Estabelecidas as três frentes do projeto e conectadas de forma coerente via Trio (SMITH. N. J., 2017) e via o compartilhamento de dados com uma instância da classe "*VariableExchange*", passou-se a validar os testes do projeto já lado a lado ao Gêmeo Digital.

Para validar o modo "*Set/Reset*" foi elaborada uma rede de Petri capaz de controlar a bancada *MPS Sorting* alocando as peças metálicas na primeira rampa, as peças vermelhas na segunda rampa e as peças pretas na terceira rampa. Esta rede de Petri pode ser conferida na figura 43.

Selecionando o arquivo XML correspondente a rede de Petri da Figura 43 na interface do programa, e iniciando a simulação, foram capturadas imagens de tela correspondentes a estados intermediários da simulação.

Figura 42 – *Teste de Conectividade OPC-UA.*

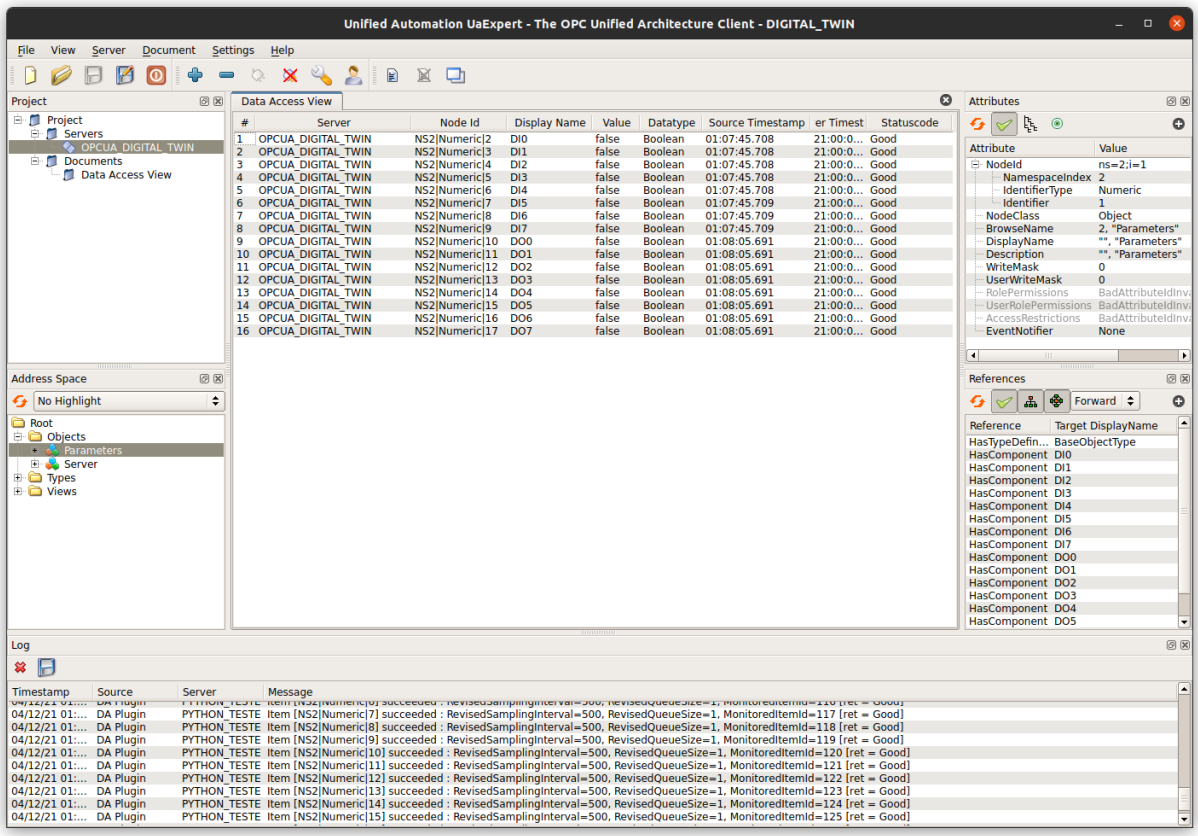
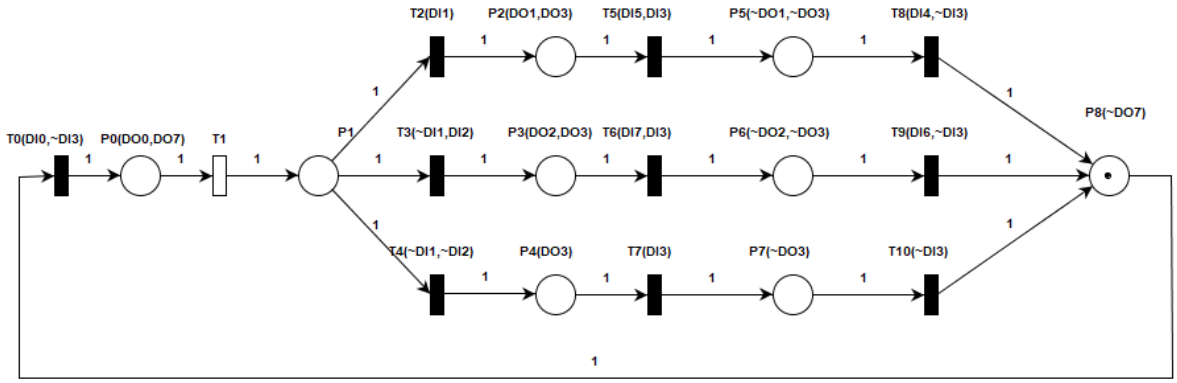
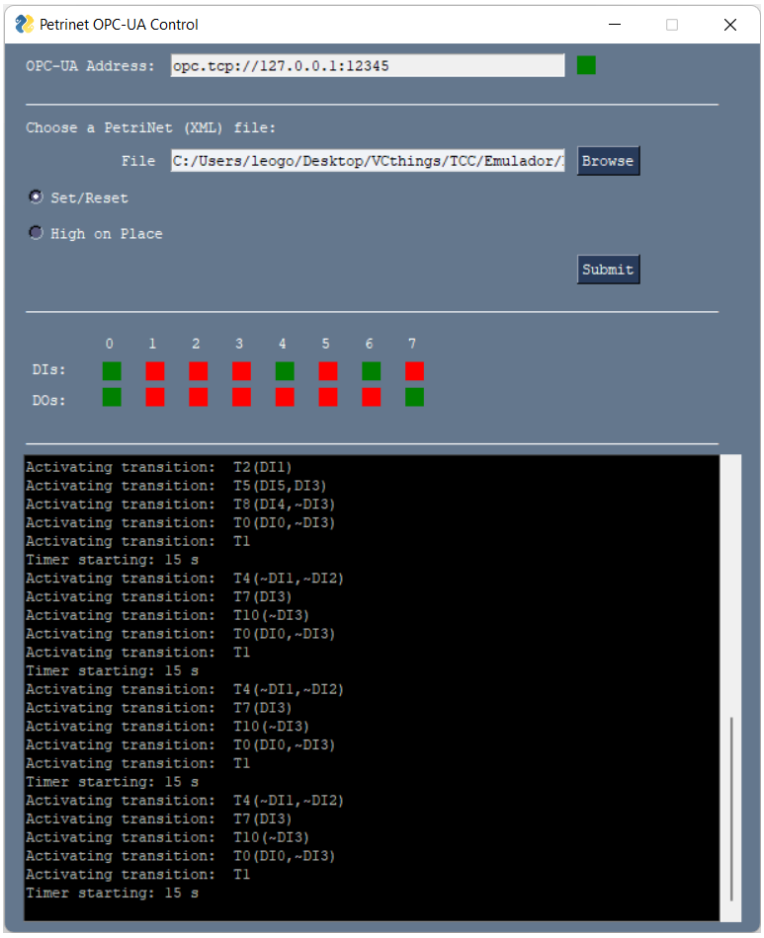


Figura 43 – *Rede de Petri - Modo Set/Reset.*



A Figura 44 representa uma dessas capturas de tela contendo o estado da simulação visto na Figura 45. É possível ver a seleção do modo "*Set/Reset*", o que exige que as saídas sejam definidas para "0" quando necessário, bem como as transições anteriores registradas na interface do usuário. As entradas DI0, DI4, e DI6, enviadas pelo Gêmeo Digital, estão em "1". A peça se encontra no começo da esteira e caminha para a posição em que os sensores definirão o tipo de peça e, com esta informação, o controlador definirá seu caminho. Nesta rede de Petri,

Figura 44 – *Interface do Usuário - Primeiro Teste - Marca em P0*



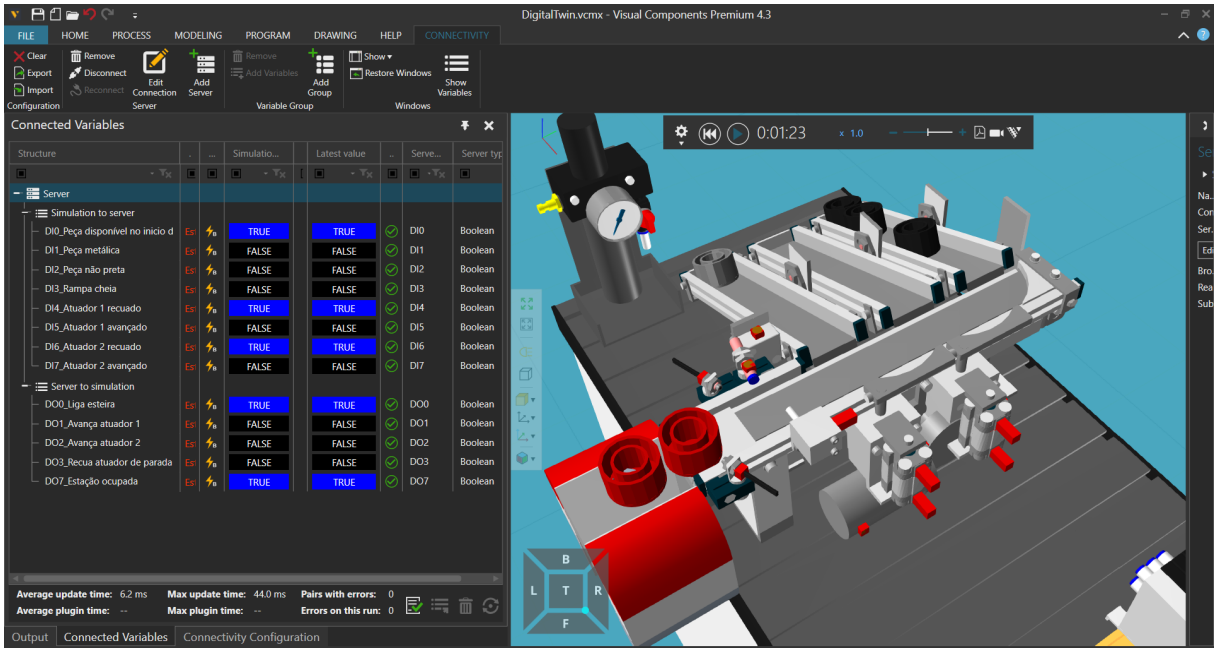
foi inserida uma temporização na transição T1 de 15 segundos, para garantir a posição da peça na região de medida dos sensores. Este tempo é mais que suficiente para que isso ocorra.

As Figuras 46 e 47 ilustram, respectivamente, a interface do usuário e o estado da simulação para um momento imediatamente posterior a transição T3 da rede de Petri da Figura 43 ser ativada. Neste caso, as entradas DI2, DI4 e DI6 estão em "1". O que indica que a peça analisada não é da cor preta e também não é metálica, portanto é vermelha, fazendo com que o programa tome a decisão de alocar a peça na segunda esteira, conforme o arquivo selecionado.

É visto nas Figuras 48 e 49 o estado descrito no Lugar P3 da rede de Petri da figura. A peça é deslocada na esteira e a ativação do sensor da rampa DI3, que indicaria a entrada da peça em uma das rampas, é aguardada para que se dispare a transição T6.

Por fim, encerrando o ciclo desta peça, nas Figuras 50 e 51, a passagem da peça pelo sensor DI3, indicando que a peça entrou de fato na rampa. Neste instante da simulação existe uma Marca em P6 na rede de Petri da Figura 43. No Gêmeo Digital, vê-se espaço disponível

Figura 45 – *Visual Components - Primeiro Teste - Marca em P0*



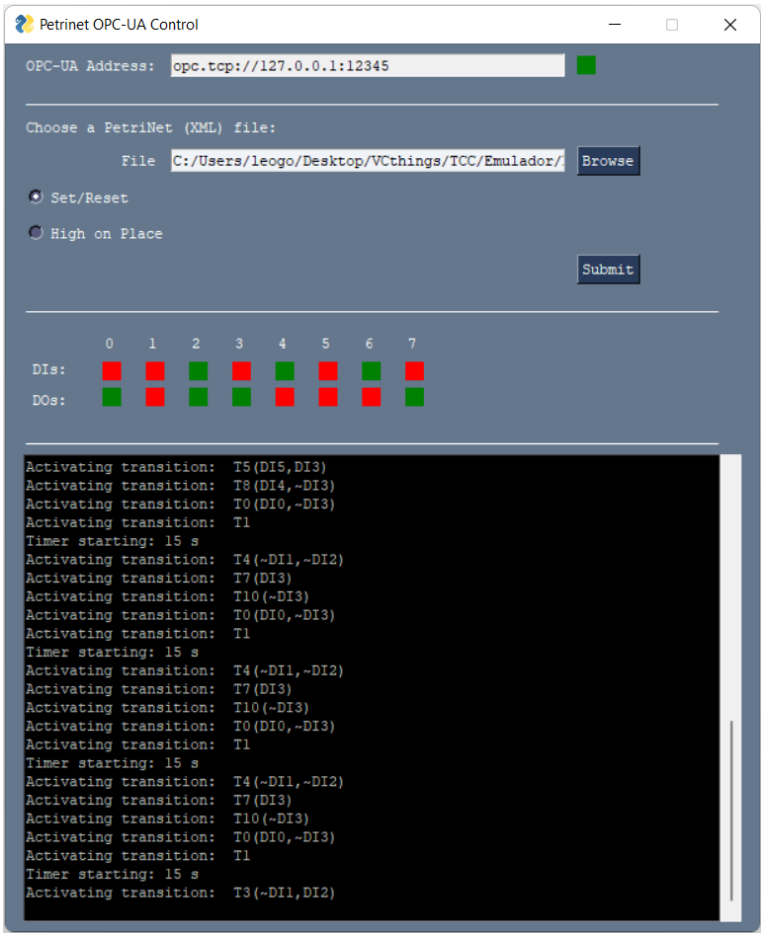
na rampa, o que indica que, após a passagem da peça pelo sensor da rampa, a peça deixará de interromper o sensor, fazendo com que DI3 seja novamente definido para "0", ativando a transição T9, para que um novo ciclo, para uma nova peça, seja iniciado. Caso a rampa esteja cheia de peças, o sensor DI3 se manteria em "1", indicando rampa cheia. Neste caso, a transição T9 não seria ativada e a estação não seria liberada para novas peças, bloqueando a simulação, como esperado.

Para validar o modo "*High on Place*", em que apenas lugares com Marcas definem suas saídas para "1", foi elaborada uma nova rede de Petri capaz de controlar a bancada MPS Sorting alocando as peças metálicas na primeira rampa, as peças vermelhas na segunda rampa e as peças pretas na terceira rampa. Esta rede de Petri pode ser conferida na Figura 52.

Foi selecionado o arquivo XML correspondente a rede de Petri da Figura 52 na interface do programa, marcando a opção "*High on Place*". Iniciando a simulação, foram capturadas imagens de tela correspondentes a estados intermediários.

Para este modo, é possível notar que, embora as ativações sejam semelhantes, a rede de Petri em sí, é bastante diferente da utilizada no modo "*Set/Reset*". Nenhum dos Lugares, na rede de Petri "*High on Place*" define as uma saída para "0". Isto ocorre pois, por padrão, apenas Lugares com Marcas definem para "1" as saídas relacionadas. Assim, para a Figura 53, vê-se que a rede de Petri foi devidamente carregada no modo "*High on Place*" e como pode ser visto na Figura 54, deu-se início a simulação. Com a ativação de T0, da Figura 52, adicionou-se

Figura 46 – *Interface do Usuário - Primeiro Teste - Marca em P3*



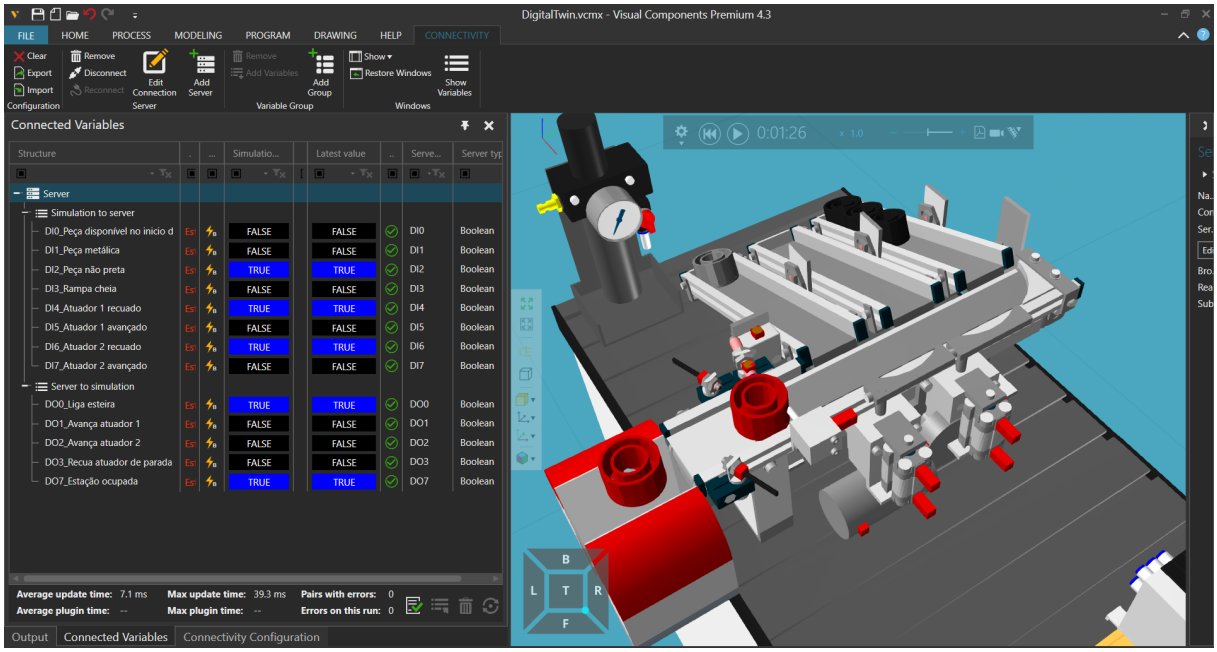
uma Marca em P2, uma Marca em P8, responsável por definir DO0 para "1" e, portanto, ligar a esteira, e uma Marca em P9, responsável por definir DO7 para "1" informando que a Estação está ocupada.

As Figuras 55 e 56, fazem referência ao estado imediatamente posterior a ativação T2, da Figura 52, neste instante, as ainda existem Marcas em P8 e P9 e, portanto, a esteira segue ligada e a Estação segue ocupada, e também existe Marca em P4, o que define para "1" as saídas DO1 e DO3, fazendo com que o primeiro atuador seja avançado e o atuador de parada no início da esteira seja recuado. Como pode ser visto na Figura 58.

Ao passar pelo sensor da rampa, a transição T5, da Figura 52, é disparada, colocando uma Marca no Lugar P7, que não está atrelado a nenhuma ação, e portanto, não redefine nenhuma saída para "1", mas redefine as saídas DO1 e DO3 para "0" ao consumir a Marca em P4. O que faz com que o primeiro atuador seja recuado e o atuador de parada no início da esteira seja avançado. Este instante pode ser visualizado nas Figuras 59 e 60.

Vê-se que, assim como no teste para o modo "*Set/Reset*", definiu-se na Figura 52 que,

Figura 47 – *Visual Components - Primeiro Test - Marca em P3*



assim que a peça deixa de interromper o sensor de rampa cheia e que é recuado o primeiro atuador (para a peça cinza) são satisfeitas as condições para o disparo da transição T8. O que completa os requisitos de Marcas para que a Transição T11 seja ativada, recomeçando o ciclo assim que uma nova peça entre na esteira.

Em (ALMIRALL; GODOY, 2021c) é disponibilizado um vídeo contendo um trecho das execuções de teste a partir da Figura 43 e em (ALMIRALL; GODOY, 2021b) é disponibilizado um vídeo contendo um trecho das execuções de teste a partir da Figura 52.

O código do interpretador de rede de Petri que estabelece servidor OPC-UA desenvolvido neste projeto pode ser encontrado em (ALMIRALL; GODOY, 2021a).

Figura 48 – *Interface do Usuário - Primeiro Teste - Marca em P3*

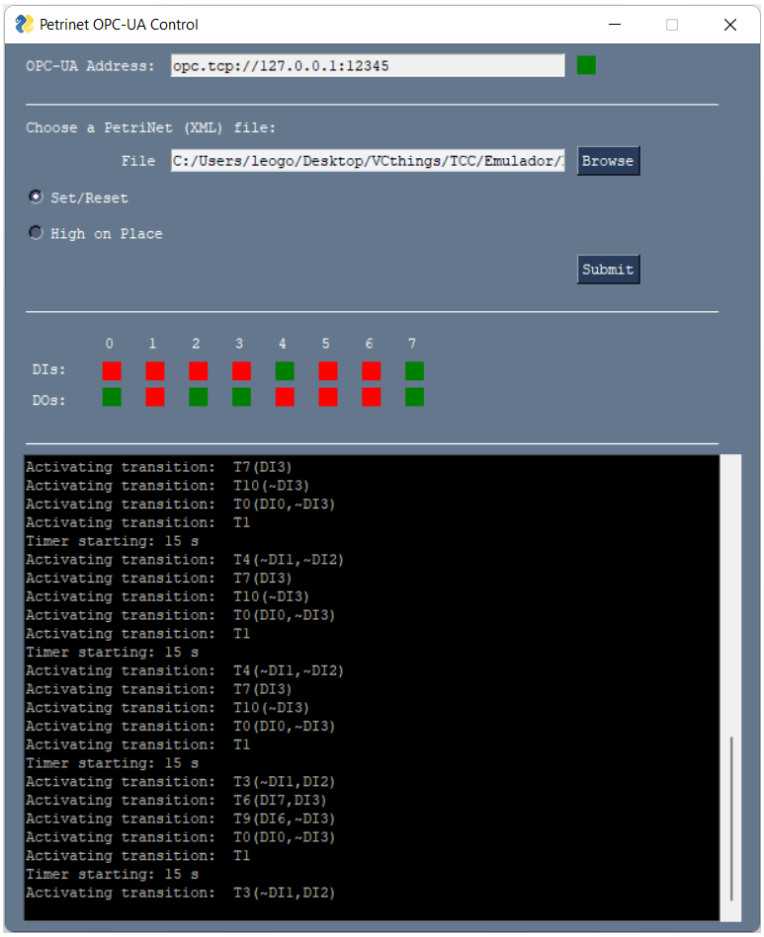


Figura 49 – *Visual Components - Primeiro Teste - Marca em P3*

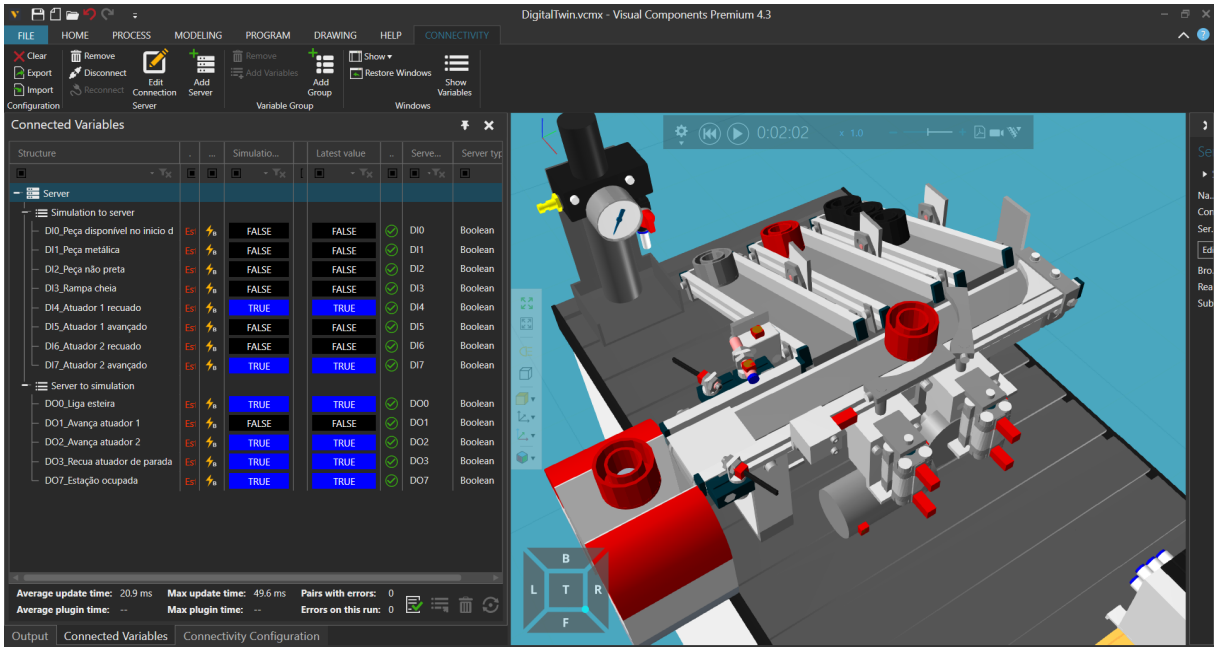


Figura 50 – *Interface do Usuário - Primeiro Teste - Marca em P6*

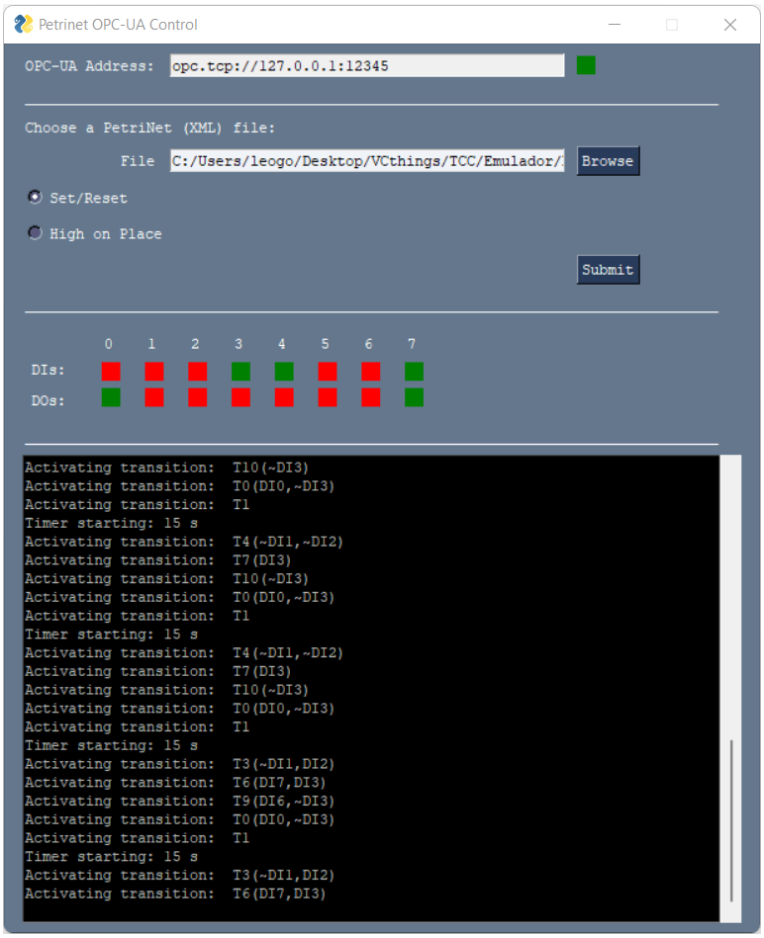


Figura 51 – *Visual Components - Primeiro Teste - Marca em P6*

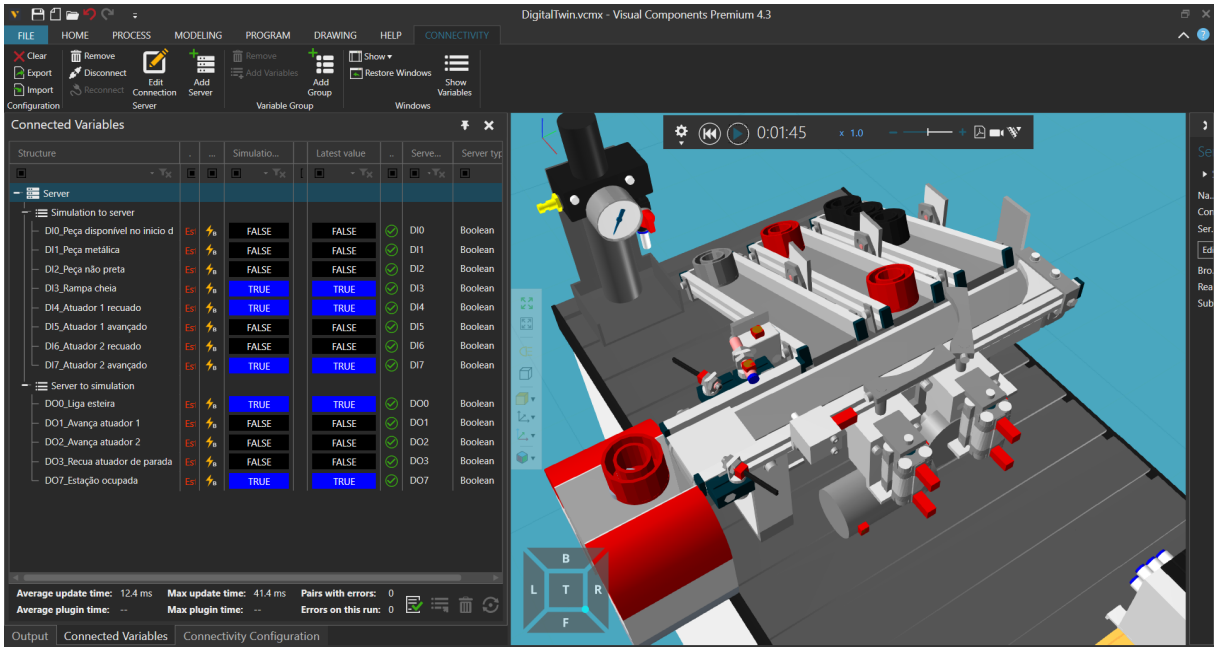


Figura 52 – *Rede de Petri - Modo High on Place.*

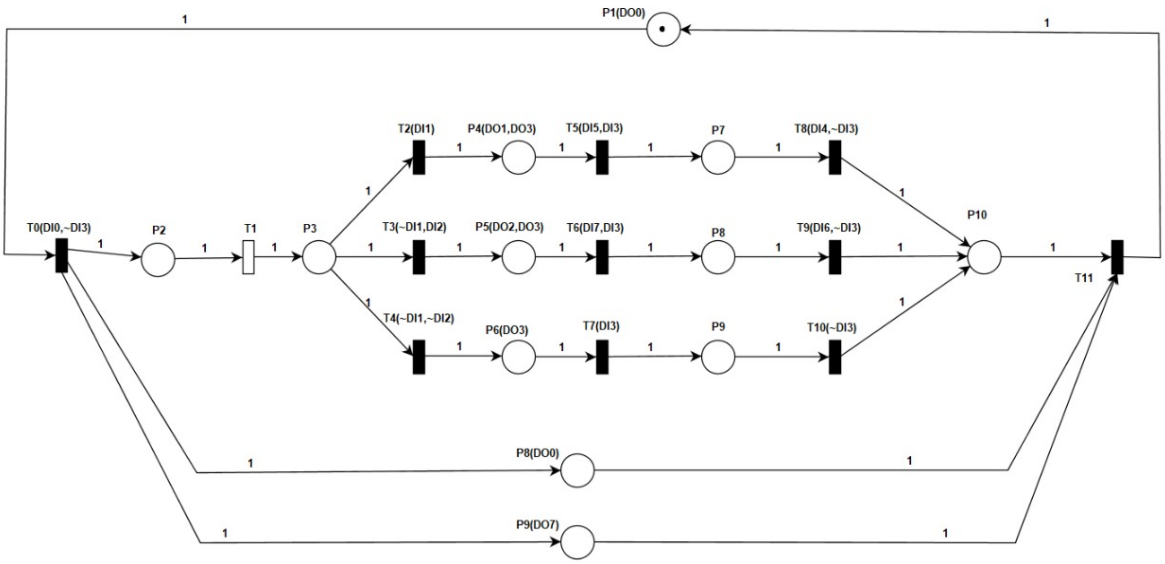


Figura 53 – *Interface do Usuário - Segundo Teste - Marcas em P2, P8 e P9*

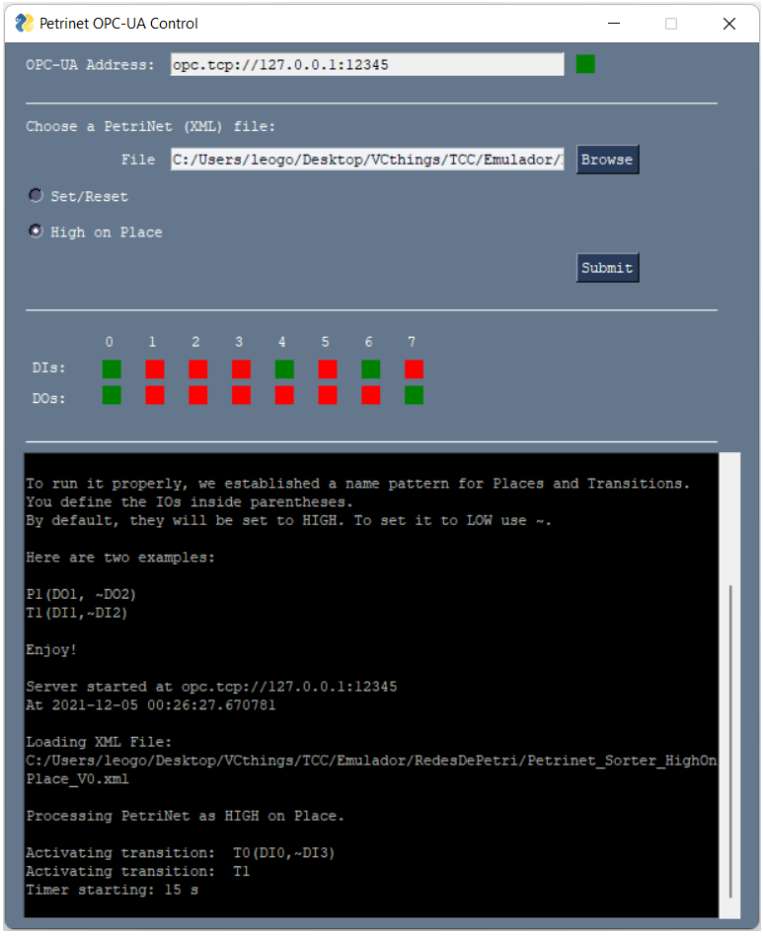


Figura 54 – *Visual Components - Segundo Teste - Marcas em P2, P8 e P9*

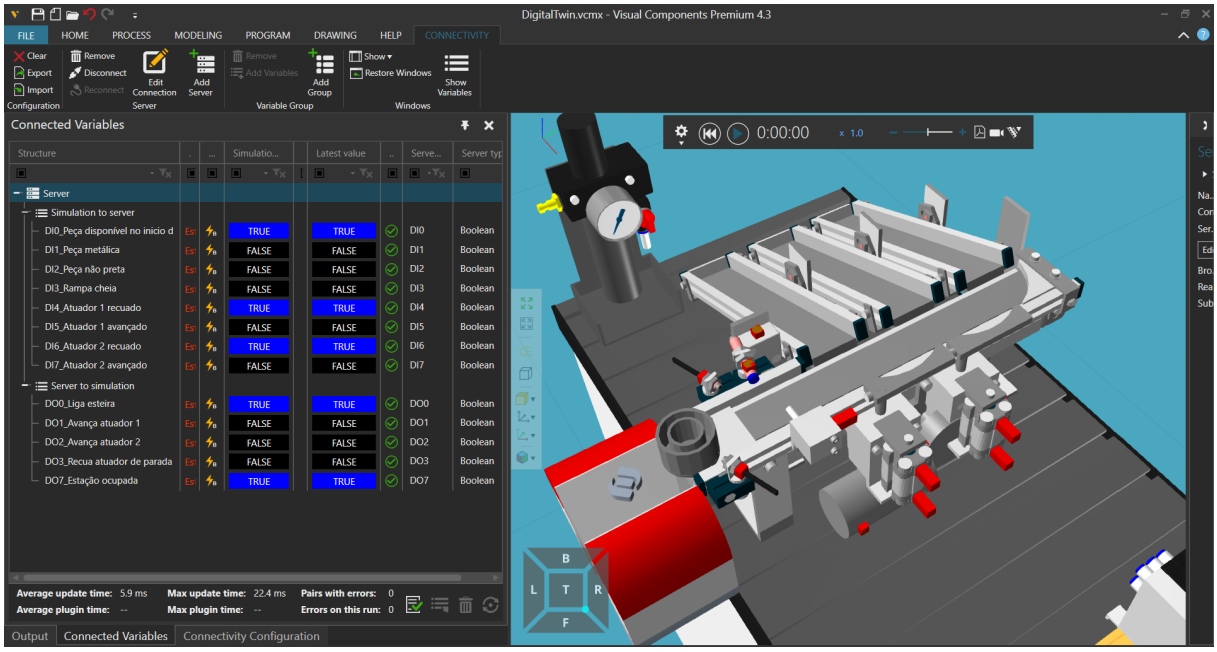


Figura 55 – *Interface do Usuário - Segundo Teste - Marcas em P4, P8 e P9*

Figura 56 – *Visual Components - Segundo Teste - Marcas em P4, P8 e P9*

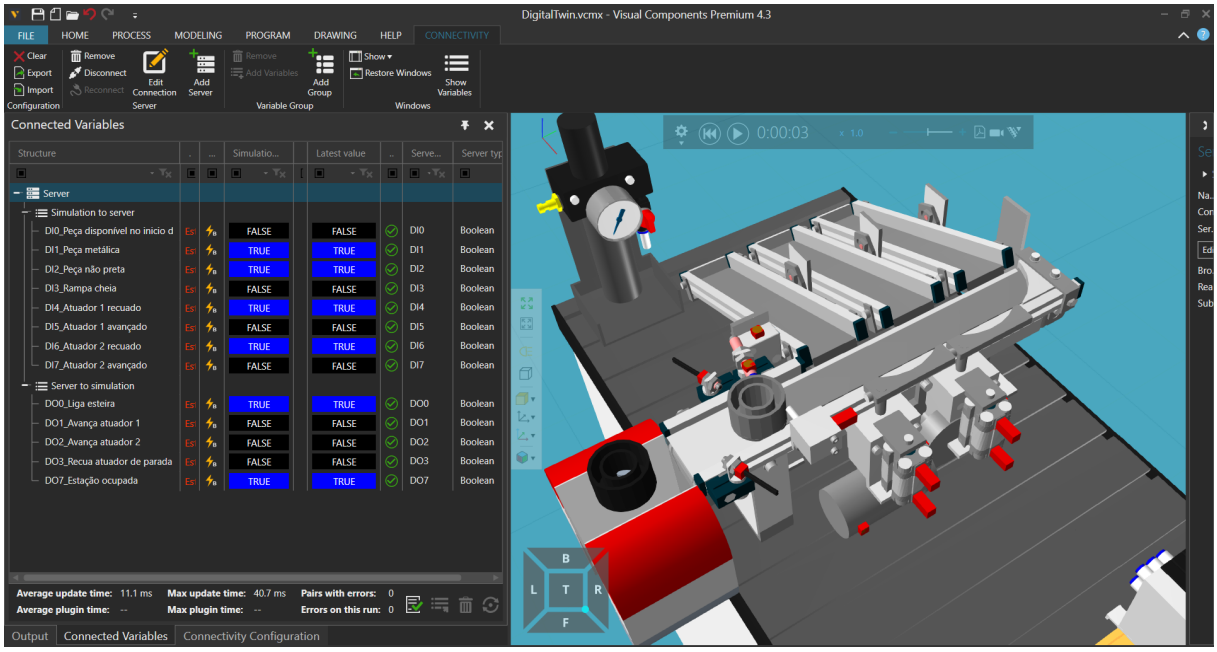


Figura 57 – *Interface do Usuário - Segundo Teste - Marcas em P4, P8 e P9*

Figura 58 – *Visual Components - Segundo Teste - Marcas em P4, P8 e P9*

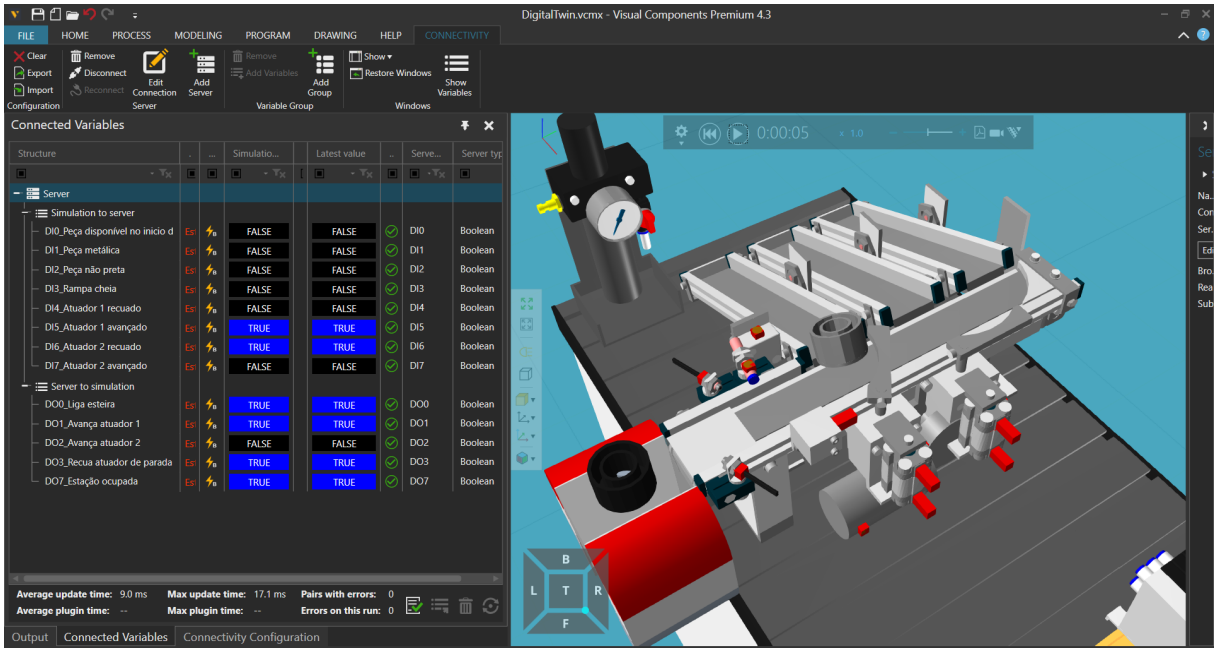


Figura 59 – *Interface do Usuário - Segundo Teste - Marcas em P7, P8 e P9*

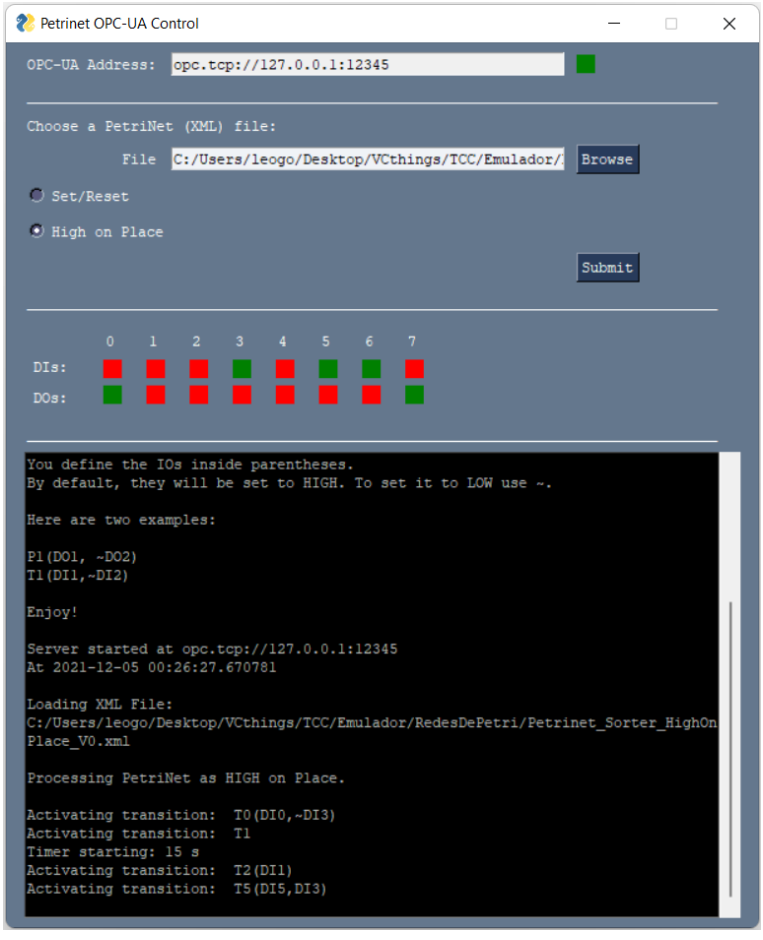
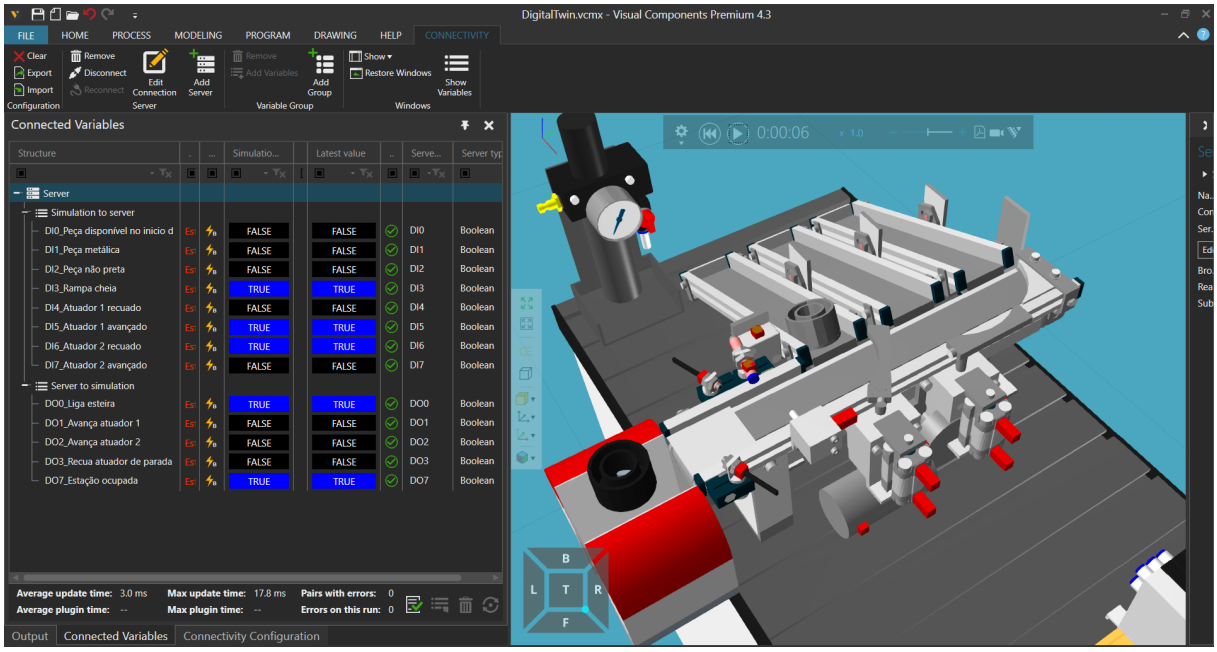


Figura 60 – *Visual Components - Segundo Teste - Marcas em P7, P8 e P9*



6 Conclusão

O trabalho concluiu com sucesso as etapas que se propôs a executar. Ao conectar um interpretador de rede de Petri, capaz de fazer a leitura de forma flexível de qualquer rede de Petri estruturada no PIPE 4.3 com um Gêmeo Digital via protocolo OPC-UA, viabilizou uma simulação fiel à realidade de uma bancada didática, permitindo testes e validações bem como o uso didático da ferramenta e a possibilidade de uso para treinamentos dos usuários, além de permitir uma forma diferente de controle da bancada, agora a partir de uma rede de Petri.

Construído com linguagem Python, abre possibilidades de extração de informações em tempo real do Gêmeo Digital em questão que podem vir a ser utilizadas para a melhora de eventuais ociosidades do processo.

O trabalho demonstra-se bastante promissor para a aplicação em sala de aula. Conforme descrito nos objetivos do projeto, abre a possibilidade de melhoria de laboratórios didáticos, principalmente envolvendo a disciplina PMR 3305 da Graduação de Engenharia Mecatrônica da Escola Politécnica, podendo ser muito utilizado em períodos de aula a distância, mas não perdendo sua relevância também em aulas presenciais, por permitir aos alunos testes diretos com redes de Petri, de forma a sintetizar a lógica de controle.

6.1 Trabalhos Futuros

É proposto como continuidade deste trabalho, a implementação de novas bancadas didáticas para ampliar as possibilidades de uso da ferramenta. Deste modo, seria possível testar as bancadas em sequência, o que permitiria estudos de melhoria do processo como um todo, evitando gargalos intermediários e aumentando o fluxo de peças entre as etapas.

Pode-se ainda, trabalhar no desenvolvimento de outros ambientes de simulação com até 8 entradas digitais e até 8 saídas digitais para o controle via rede de Petri por intermédio do Servidor OPC-UA estabelecido por este projeto. Seria possível desta forma, controlar sistemas diferentes ao das bancadas didáticas, sendo possível inclusive testar variados sistemas a eventos discretos.

Sugere-se o uso e a avaliação da funcionalidade da ferramenta na bancada real, neste caso, tanto com a bancada *MPS-Sorting* Festo quanto com outras bancadas didáticas, a fim de validar a conectividade OPC-UA com tal sistema real.

Por fim, propõe-se a aplicação de forma didática da ferramenta, a fim de fornecer uma noção mais paupável e intuitiva da funcionalidade de redes de Petri em sistemas reais e simulados. Para a aplicação didática, seria interessante a condução de um estudo a fim de avaliar os ganhos do uso do programa em sala de aula como ferramenta de fixação para testes de redes de Petri.

Referências

ALMIRALL, F.; GODOY, L. **Código em Python de Programa. Projeto E Integração de Gêmeo Digital de bancada didática e emulador de rede de Petri**. 2021. Disponível em: <https://github.com/FelipeAlmirall/Petrinet_Emulator_OPC-UA>. Acesso em: 05 de dezembro de 2021. Citado na página 63.

ALMIRALL, F.; GODOY, L. **Teste de execução - HighOnPlace: Gêmeo Digital controlado por emulador de Rede de Petri**. 2021. Disponível em: <https://www.youtube.com/watch?v=U74VcixBdAUm>. Acesso em: 05 de dezembro de 2021. Citado na página 63.

ALMIRALL, F.; GODOY, L. **Teste de execução - Set/Reset: Gêmeo Digital controlado por emulador de Rede de Petri**. 2021. Disponível em: <https://www.youtube.com/watch?v=vqvN9E6n3-8>. Acesso em: 05 de dezembro de 2021. Citado na página 63.

BABU, V.; NICOL, D. Emulation/simulation of plc networks with the s3f network simulator. **2016 Winter Simulation Conference (WSC)**, p. 1475–1486, 12 2016. Citado na página 7.

BAHRIN, M. et al. Industry 4.0: A review on industrial automation and robotic. **Jurnal Teknologi**, v. 78, 06 2016. Citado na página 1.

BARROS, J. P. Modularidade em redes de petri. 2006. Disponível em: <http://hdl.handle.net/10400.26/1085>. Citado na página 7.

BERISHA, A.; CARUSO, C.; HARTEIS, C. The concept of a digital twin and its potential for learning organizations. **Digital Transformation of Learning Organizations**, p. 95–114, 01 2021. Citado na página 4.

CHEN, M. Biopnml. 2002. Disponível em: <https://www.techfak.uni-bielefeld.de/~mchen/BioPNML/Intro/pnfaq.html>. Citado 3 vezes nas páginas 11, 12 e 13.

EL-GENK, M. et al. Implementation and validation of plc emulation and data transfer. 2019. Citado na página 6.

FESTO. **MPS Festo Sorting Station**. 2015. Disponível em: <https://www.festo-didactic.com/ov3/media/customers/1100/8046325_manual_de_en_es_fr_1.pdf>. Acesso em: 2 de Agosto de 2021. Citado 2 vezes nas páginas 16 e 17.

FESTO. **CIROS**. 2021. Disponível em: <https://www.festo-didactic.com/int-en/services/printed-media/manuals/ciros.htm>. Acesso em: 10 de maio de 2021. Citado na página 9.

freeOPCUA. **OPCUA Python Library**. 2020. Disponível em: <https://freeopcua.github.io/>. Citado na página 56.

GöKALP, M. et al. Big data for industry 4.0: A conceptual framework. **2016 International Conference on Computational Science and Computational Intelligence (CSCI)**, p. 431–434, 12 2016. Citado na página 5.

II, J. Introduction to modeling and simulation. **29th conference on Winter simulation**, p. 9–16, 01 2004. Citado na página 1.

KAARLELA, T.; PIESKä, S.; PITKäAHO, T. Digital twin and virtual reality for safety training. In: **2020 11th IEEE International Conference on Cognitive Infocommunications (CogInfoCom)**. [S.l.: s.n.], 2020. p. 000115–000120. Citado na página 6.

LASI, H. et al. Industry 4.0. **Business & Information Systems Engineering**, v. 6, n. 4, p. 239–242, Aug 2014. ISSN 1867-0202. Disponível em: <https://doi.org/10.1007/s12599-014-0334-4>. Citado na página 1.

LE, Q. T.; PEDRO, A.; PARK, C. S. A social virtual reality based construction safety education system for experiential learning. In: . [s.n.], 2015. v. 79, n. 3, p. 487–506. ISSN 1573-0409. Disponível em: <https://doi.org/10.1007/s10846-014-0112-z>. Citado na página 6.

MIYAGI, P. E. Fundamentos do controle de sistemas a eventos discretos. In: **Controle Programável**. [S.l.: s.n.], 1996. Citado na página 11.

NIKOLAEV, S. et al. Implementation of "digital twin" concept for modern project-based engineering education. In: CHIABERT, P. et al. (Ed.). **Product Lifecycle Management to Support Industry 4.0**. Cham: [s.n.], 2018. p. 193–203. ISBN 978-3-030-01614-2. Citado na página 2.

OPC-UA. **OPC Unified Architecture**. 2021. Disponível em: <https://opcfoundation.org/about/opc-technologies/opc-ua/>. Acesso em: 10 de maio de 2021. Citado na página 10.

OSBORNE, M.; MAVERS, S. Integrating augmented reality in training and industrial applications. **2019 Eighth International Conference on Educational Innovation through Technology (EITT)**, p. 142–146, 10 2019. Citado na página 4.

OSMAN, M. S.; BAKAR, B. B. A. Emulation of petri net controller with programmable logic controller in industrial process plant. In: **2014 4th International Conference on Engineering Technology and Technopreneuship (ICE2T)**. [S.l.: s.n.], 2014. p. 272–277. Citado 2 vezes nas páginas 8 e 9.

PANETTA, K. **Gartner.com**. 2016. Disponível em: <https://www.gartner.com/smarterwithgartner/gartners-top-10-technology-trends-2017>. Acesso em: 7 de maio de 2021. Citado na página 4.

Platform Independent Petri net Editor. **Pipe**. 2009. Disponível em: <http://pipe2.sourceforge.net/>. Citado 2 vezes nas páginas 26 e 53.

POLA, D. F. Treinamento: Didactic - mps modular production system. In: **Treinamento: Didactic - MPS Modular Production System**. [S.l.: s.n.], 2013. Citado 2 vezes nas páginas 14 e 25.

PySimpleGUI. **PySimpleGUI Python Library**. 2020. Disponível em: <https://pysimplegui.readthedocs.io/>. Citado na página 52.

QUEZADA, J. C. et al. Simulation and validation of diagram ladder—petri nets. **The International Journal of Advanced Manufacturing Technology**, v. 88, n. 5, p. 1393–1405, Feb 2017. ISSN 1433-3015. Disponível em: <https://doi.org/10.1007/s00170-016-8638-9>. Citado na página 8.

QUINALHA, E. Gêmeos digitais, o futuro da indústria 4.0: estudo de caso. p. 60, 2018. Citado na página 1.

RESEARCH, I. of R. **Short introduction into 3D Simulation and Offline Programming of robot-based workcells with COSIMIR**. 2000. Disponível em: <http://www.int76.ru/upload/iblock/74c/74caae11f575cafdc754991bc9f43d67.pdf>. Acesso em: 9 de maio de 2021. Citado na página 9.

SEPASGOZAR, S. M. Digital twin and web-based virtual gaming technologies for online education: A case of construction management and engineering. **Applied Sciences**, v. 10, n. 13, 2020. ISSN 2076-3417. Disponível em: <https://www.mdpi.com/2076-3417/10/13/4678>. Citado 2 vezes nas páginas 2 e 5.

SIEMENS. **Smart. Semantic. OPC UA – structured data up to the cloud.** 2021. Disponível em: <https://opcfoundation.org/about/opc-technologies/opc-ua/>. Acesso em: 10 de maio de 2021. Citado na página 10.

SILVIA, C. The impact of simulations on higher level learning. **Journal of Public Affairs Education**, 01 2012. Citado na página 5.

SIM, Z. H. et al. Design of virtual reality simulation-based safety training workshop. In: **2019 IEEE International Symposium on Haptic, Audio and Visual Environments and Games (HAVE)**. [S.l.: s.n.], 2019. p. 1–6. Citado na página 6.

SMITH. N. J. **Trio Python Library**. 2017. Disponível em: <https://trio.readthedocs.io/en/stable/>. Citado 2 vezes nas páginas 51 e 58.

UA-Expert. **UA-Expert**. 2018. Disponível em: <https://www.unified-automation.com/>. Citado na página 58.

VAIDYA, S.; AMBAD, P.; BHOSLE, S. Industry 4.0 – a glimpse. **Elsevier B.V**, v. 20, p. 233–238, 01 2018. Citado na página 1.

Visual Components. **Visual Components 4.3**. 2021. Disponível em: <https://www.visualcomponents.com/>. Citado 5 vezes nas páginas 19, 20, 21, 22 e 51.