FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Framework para Digital Twins de Processos

Fernando Jorge Pinheiro da Silva

MESTRADO INTEGRADO EM ENGENHARIA ELETROTÉCNICA E DE COMPUTADORES

Orientador: Jorge Pinho de Sousa

Co-orientador: João Pedro Tavares Vieira Basto

28 de junho de 2021

Resumo

O acelerado e constante desenvolvimento tecnológico tem contribuído cada vez mais para a criação de novas soluções no ramo da automação. Com o surgir do conceito da Indústria 4.0, as empresas procuram fazer parte da vanguarda desta revolução, no âmbito de assegurar qualquer vantagem competitiva. O *Digital Twin* (DT), como pilar da Indústria 4.0, revela-se crucial na implementação deste conceito e tem ganho uma grande popularidade na indústria transformadora.

Nos dias de hoje, é imperativo aos gestores de produção compreender o estado atual - em tempo real - dos seus recursos operacionais. Neste sentido, os *Digital Twins* de processos procuram replicar o comportamento de processos de sistemas reais, a qualquer momento. Dado que grande parte dos DT desenvolvidos são projetados para cenários específicos, surge o interesse em criar uma *framework* que seja flexível e adaptável a qualquer tipo de processos, com funcionalidades que acrescentem valor ao DT.

A presente dissertação apresenta uma *framework* para DT de processos que permite monitorizar em tempo real o *shop floor*, fazer a simulação do futuro a partir de planos de produção e comunicar com serviços inteligentes externos. Efetivamente, a monitorização permite a visualização 3D dos eventos que estão a ocorrer na fábrica, no que se trata do DT em particular. O modo de simulação consegue simular o tráfego no *shop floor* dado um plano de produção introduzido pelo utilizador. Por fim, a comunicação com serviços inteligentes reforça a interoperabilidade da *framework*, sendo que neste projeto se recorreu a um algoritmo externo que otimiza o plano de produção do utilizador, cujos resultados do *makespan* são comparados com soluções não otimizadas. Estas três componentes são a base da plataforma desenvolvida com que se procura potenciar as capacidades dos DT de processos.

Abstract

The accelerated and constant technological development is increasingly contributing to the creation of new solutions in the field of automation. With the emergence of the Industry 4.0 concept, companies are seeking to be at the forefront of this revolution in order to secure any competitive advantage. The Digital Twin (DT), as one of the pillars of Industry 4.0, proves to be crucial in the implementation of this concept and is gaining popularity among the manufacturing industry.

Nowadays, it is imperative for production managers to understand the current state - in real time - of their operational resources. In this sense, process DTs seek to replicate the behavior of real system processes at any given time. Given that most of the developed DTs are designed for specific scenarios, the interest in creating a framework that is flexible and adaptable to any kind of process, with functionalities that add value to the DT, arises.

This dissertation presents a framework for process DTs that allows real-time monitoring of the shop floor, simulation of the future based on production plans and communication with external intelligent services. Effectively, monitoring allows for 3D visualization of the events that are occurring in the plant, with regards to the DT in particular. Simulation mode can simulate the traffic on the shop floor given a production plan submitted by the user. Finally, communication with intelligent services reinforces the interoperability of the framework, and in this project an external algorithm was used to optimize the user's production plan, whose makespan results are compared with non-optimized solutions. These 3 components are the basis of the developed platform with which we seek to leverage the capabilities of process DTs.

Agradecimentos

Esta dissertação marca o fim de uma jornada recheada de alegrias, obstáculos e desafios. Marca a realização de um objetivo na minha vida e abre as portas para um novo futuro. Não fosse a contribuição de várias pessoas, jamais seria possível viver este momento.

Em primeiro lugar, deixo o meu agradecimento ao meu orientador, Professor Doutor Jorge Pinho de Sousa, e ao meu co-orientador, Professor João Pedro Basto, pela oportunidade, pelo acompanhamento constante na realização do projeto e pelo fantástico apoio que sempre me deram, sempre preparados para me auxiliar nas maiores dificuldades. Quero também agradecer ao Engenheiro Romão Santos e ao Engenheiro Bruno Cunha pela disponibilidade e ajuda no desenvolvimento do trabalho.

Deixo o meu agradecimento ao INESC TEC - Instituto de Engenharia de Sistemas e Computadores, Tecnologia e Ciência - e a todo o pessoal pelos recursos e, apesar de não ter sido possível usufruir das mesmas devido à situação pandémica que se vive, pelas instalações cedidas.

Aos meus pais, por todo o carinho e por me proporcionarem a oportunidade de chegar aqui. Obrigado por confiarem sempre em mim e por serem um dos meus pilares. Não há palavras que descrevam a gratidão que sinto por vocês.

À minha namorada, pelo amor, pela amizade e pelo apoio incondicional. A ti, Catarina, minha cara metade, melhor amiga e companheira de sempre, obrigado por me acompanhares desde o início ao fim deste ciclo. Obrigado por toda a ajuda e motivação.

Por fim, agradeço a toda a minha família, desde a família Silva à família Pinheiro, e aos meus amigos da FEUP e aos velhos amigos de Alpendorada pela força e incentivo. Um sincero obrigado a todos vós.

Fernando Silva

"The present is theirs;
the future, for which I really worked, is mine"

Nikola Tesla

Conteúdo

Lista de Figuras

Lista de Tabelas

Abreviaturas e Símbolos

3D	Tridimensional
AGV	Automated Guided Vehicle
ARPANET	Advanced Research Projects Agency Network
CAD	Computer Aided Design
CNC	Computer Numerical Control
CPS	Cyber-Physical System
CSV	Comma Separated Values
BD	Base de Dados
DT	Digital Twin
ID	Identificador
INESC TEC	Instituto de Engenharia de Sistemas e Computadores, Tecnologia e Ciência
IoS	Internet of Services
IoT	Internet of Things
IP	Internet Protocol
JSON	JavaScript Object Notation
KPI	Key Performance Indicator
MES	Manufacturing Execution System
MQTT	Message Queuing Telemetry Transport
NASA	National Aeronautics and Space Administration
NSF	National Science Foundation
PLC	Programmable Logic Controller
P&G	Procter & Gamble
RFID	Radio-Frequency Identification
TCP	Transmission Control Protocol
WS	Workstation

Capítulo 1

Introdução

1.1 Contexto e motivação

A introdução da era da indústria 4.0 deu vida a diversos novos conceitos que suportam a implementação da mesma. Um desses é o *Digital Twin*, que tal como o seu nome indica pode ser descrito como uma cópia digital de um objeto, de um processo ou até mesmo de um sistema. Este termo tem vindo a ganhar volume com o evoluir da tecnologia, mas já vão mais cerca de 50 anos desde os primeiros vestígios da sua presença.

"Houston, we've had a problem" [1], ouvia-se a 2 de Abril de 1970. Dava-se uma explosão nos tanques de oxigénio do voo espacial tripulado Apollo 13 e instalava-se a incerteza no espaço e na Terra. A solução, no entanto, estava nesta última. No planeta Terra encontrava-se uma cópia digital da nave espacial que se encontrava a mais de 300 mil de quilómetros de distância, no que terá sido um dos primeiros momentos de aplicação real de um *Digital Twin* - na altura apenas um conjunto de simuladores controlados por redes de computadores. Deste modo, foi possível estudar e testar vários cenários e decisões a tomar no salvamento da tripulação, sem implicar a vida destes na equação. O risco tornou-se um risco calculado com a possibilidade de simulação fornecida pelo *Digital Twin* desenvolvido pela NASA, tornando este episódio no que é hoje conhecido como um "falhanço bem sucedido", diz Filomena Naves [2]. Surge assim um dos primeiros sucessos com *Digital Twins*.

Efetivamente, a complexidade crescente dos processos na indústria transformadora exige da mesma forma um cálculo do risco, uma simulação do presente e uma previsão do futuro. O *Digital Twin* permite tudo isto: desde a conceção de designs a implementar, passando pela análise *real-time* dos KPI, à simulação com alterações de layouts, este conceito dá força à fusão do mundo real com o mundo virtual, que até hoje é um dos grandes *bottlenecks* ao sucesso do *smart manufacturing*.

> "A Digital Twin is an instance of a digital model that represents an entity, which includes assets, processes, and systems which are sufficient to meet the requirements of a set of use cases." Pieter van Schalkwyk [3]

Com o constante evoluir da tecnologia, a aplicação de um *Digital Twin* na indústria transformadora pode traduzir-se numa vantagem competitiva para qualquer empresa. A capacidade de visualizar em tempo real o estado dos constituintes de uma fábrica aliada ao poder de simulação de diversos cenários, faz do *Digital Twin* uma poderosa ferramenta. As ferramentas gráficas do *Digital Twin* a desenvolver, em conjunto com a facilidade em passar do estado atual a hipotéticos cenários futuros através da simulação, possibilitam quase que "andar" pela fábrica e testar diversas mudanças a aplicar, tornando a compreensão do estado desta mais intuitiva. Adicionalmente, a monitorização constante do desempenho no *shop floor* contribui para um aumento significativo da qualidade do produto. Na verdade, estas características são também responsáveis por grandes reduções no número de avarias e de acidentes, potencialmente resultando numa grande queda da carga económica que são as manutenções e reparações.

Em suma, sendo o *Digital Twin* um dos protagonistas da revolução da indústria transformadora com a indústria 4.0, surge o interesse na implementação prática deste conceito que certamente fará parte do futuro dos grandes fabricantes.

1.2 Objetivos da dissertação

O grande objetivo desta dissertação consiste no planeamento e desenvolvimento de uma *framework* para *Digital Twins* de processos com visualização 3D de um *shop floor*, a ser usado por gestores de produção e de instalações para um melhor entendimento em *real-time* do estado dos seus recursos operacionais e processos. Para além de um modelo visual, esta base do *Digital Twin* fornece ao utilizador informações em tempo real do estado de todos os elementos espalhados pelo *shop floor*, permitindo também a simulação do comportamento da fábrica a partir de planos de produção e a comunicação com outros serviços inteligentes.

Inicialmente, o objetivo passa por criar uma estrutura de dados que possibilite representar a informação trocada entre o *shop floor* e o *Digital Twin*. Depois serão considerados os conceitos IoT para a comunicação dos sensores e atuadores com o *Digital Twin*. Este receberá os sinais para mapeamento dos estados para a adaptação do modelo digital em tempo real. Com isto, será construído o modelo 3D virtual do *shop floor* no software *FlexSim*.

Posteriormente, o utilizador poderá introduzir os seus planos de produção e simular o tráfego na sua fábrica. Isto facilita a análise de diferentes combinações de planos de produção em paralelo com a monitorização em tempo real.

Finalmente, a plataforma deve ser dotada de uma capacidade de comunicar com serviços externos. Neste projeto em particular, procura-se aceder a um serviço de otimização que redefine a sequência de produção do plano introduzido, com o objetivo de minimizar o *makespan*.

1.3 Metodologia

Esta dissertação segue uma metodologia híbrida que junta técnicas baseadas em *Digital Twins* e de simulação, com a finalidade de se criar uma *framework* que promove a monitorização em

tempo real dos processos num *shop floor*, aliada à simulação de planos de produção e que permite a comunicação com serviços inteligentes externos.

Inicialmente, deve ser desenvolvido o módulo base, a monitorização em tempo real. Este será o espelho de um *shop floor* que se ajusta em tempo real conforme as alterações no sistema físico, cujo comportamento é emulado por um ficheiro CSV. Nesta primeira fase, o *Digital Twin* constitui um modelo 3D, uma base de dados e uma aplicação que faz o processamento das mensagens provenientes do sistema real. É também adaptável a diferentes tipos de produção, sendo esta flexibilidade uma das principais características da plataforma.

Seguidamente, será implementado o modo de simulação que permite ao utilizador introduzir o seu plano de produção, que é simulado no modelo visual tanto de forma isolada, como em paralelo com o último *snapshot* da monitorização. Para além da capacidade de visualização, este modo concretizado quase exclusivamente no *software FlexSim* fornece também informações relevantes associadas às diferentes combinações de planos que possam ser feitas, de que é exemplo o *makespan*.

Por fim, a plataforma deve ser capaz de comunicar com serviços externos, isto é, serviços que não fazem parte da fábrica ou da *framework* a desenvolver. Nesta dissertação comunicou-se com um programa de otimização que, a partir do plano introduzido pelo utilizador, pudesse encontrar a melhor solução para a sequência de produção no âmbito de aumentar a produtividade, minimizando os *makespans*. Neste contexto, o aumento da produtividade refere-se à capacidade de produzir o mesmo número de peças num período menor, alcançado apenas por uma alteração na sequência de produção.

A Figura 1.1 apresenta um esquema com as fases supramencionadas.

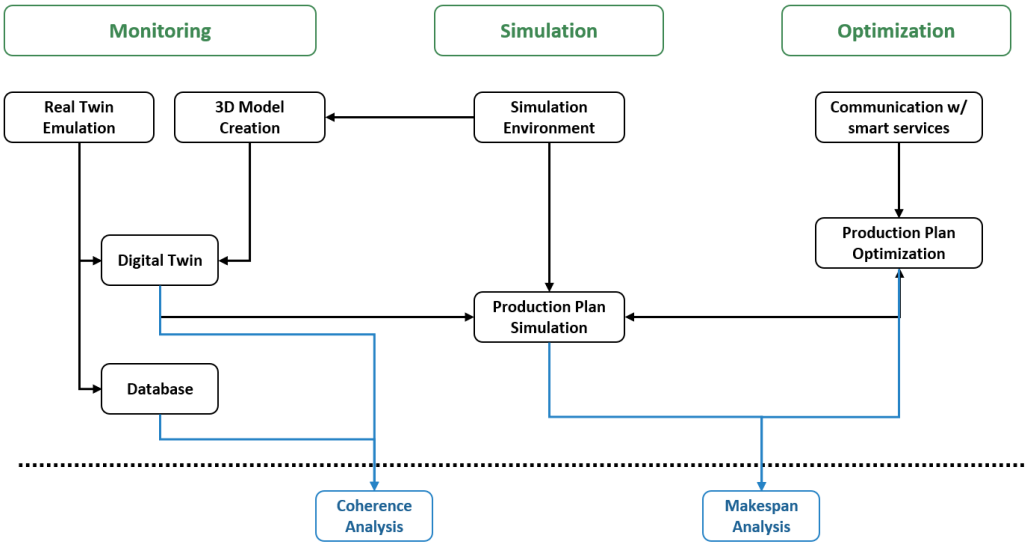

Figura 1.1: Método de abordagem

1.4 Estrutura da dissertação

A estrutura da dissertação divide-se em sete capítulos, sendo os primeiros dois de índole introdutória ao trabalho a desenvolver e ao contexto em que este se enquadra, os quatro seguintes relativos à explanação do trabalho desenvolvido em concreto e o último de cariz conclusivo.

No Capítulo 1, é definido o contexto, a motivação e os objetivos deste trabalho e é apresentado o método de abordagem e a estrutura da dissertação.

O Capítulo 2 compõe a revisão bibliográfica relativa aos temas pertinentes para a concretização deste trabalho. São estudados alguns conceitos em que se insere o *Digital Twin*, como é exemplo a Indústria 4.0 ou os sistemas cíber-físicos, e são analisados de forma mais detalhada os tipos e a classificação de *Digital Twins*.

No Capítulo 3, é descrito o caso real em análise e as respetivas características. Aqui é apresentada a arquitetura dimensionada para concretização da *framework*.

Nos Capítulos 4, 5 e 6 é abordado o trabalho desenvolvido no que diz respeito à monitorização em tempo real, à simulação de planos de produção e à otimização dos mesmos planos, respetivamente.

Por último, no Capítulo 7, descrevem-se as conclusões do trabalho conseguido e discutem-se potenciais implementações que possam ser interessantes efetivar como trabalho futuro.

Capítulo 2

Revisão Bibliográfica

Neste capítulo serão abordados alguns tópicos cruciais ao entendimento do conceito e da implementação de um *Digital Twin* (DT). Estes são resultado de um trabalho de pesquisa bibliográfica, cujas publicações são devidamente referenciadas.

Inicialmente será feita uma breve abordagem à Indústria 4.0, considerando a sua relevância com o tema em estudo. Dado que o DT se encaixa perfeitamente nas tecnologias deste conceito, é feito um estudo à sua evolução e aos princípios de *design* a si associados.

Posteriormente, serão apresentados os sistemas ciber-físicos (CPS) e a Internet das Coisas (IoT), termos diretamente ligados à Indústria 4.0 e que se enquadram na implementação do DT. Os CPSs são caracterizados pela sua capacidade em tratar e transmitir informação e em criar réplicas digitais de processos em tempo real num ambiente digital [4], enquanto que a *Internet of Things* promove a conectividade e comunicação entre os todos os elementos constituintes de uma fábrica.

Por fim, será aprofundado o conceito *Digital Twin*. Nesta secção é estudada a origem e evolução do termo, os níveis de integração, os tipos de classificação e aplicação e arquiteturas desenvolvidas para a implementação de DTs.

2.1 Indústria 4.0

2.1.1 Origem e evolução do conceito

Depois da primeira, segunda e terceira revoluções industriais que levaram a mudanças de paradigma na produção, chega a apelidada quarta revolução industrial, a Indústria 4.0. Nesta ambiciona-se a produção inteligente, isto é, automatizar os processos e fluxos de informação numa fábrica com recurso a tecnologias e conceitos como, entre outros, o Sistema Ciber-Físico, a Internet das Coisas, o *Big Data* e a Computação em Nuvem - *Cloud Computing*. Efetivamente, do mesmo modo que as revoluções industriais prévias, a Indústria 4.0 tem origem nas inovações tecnológicas mais recentes.

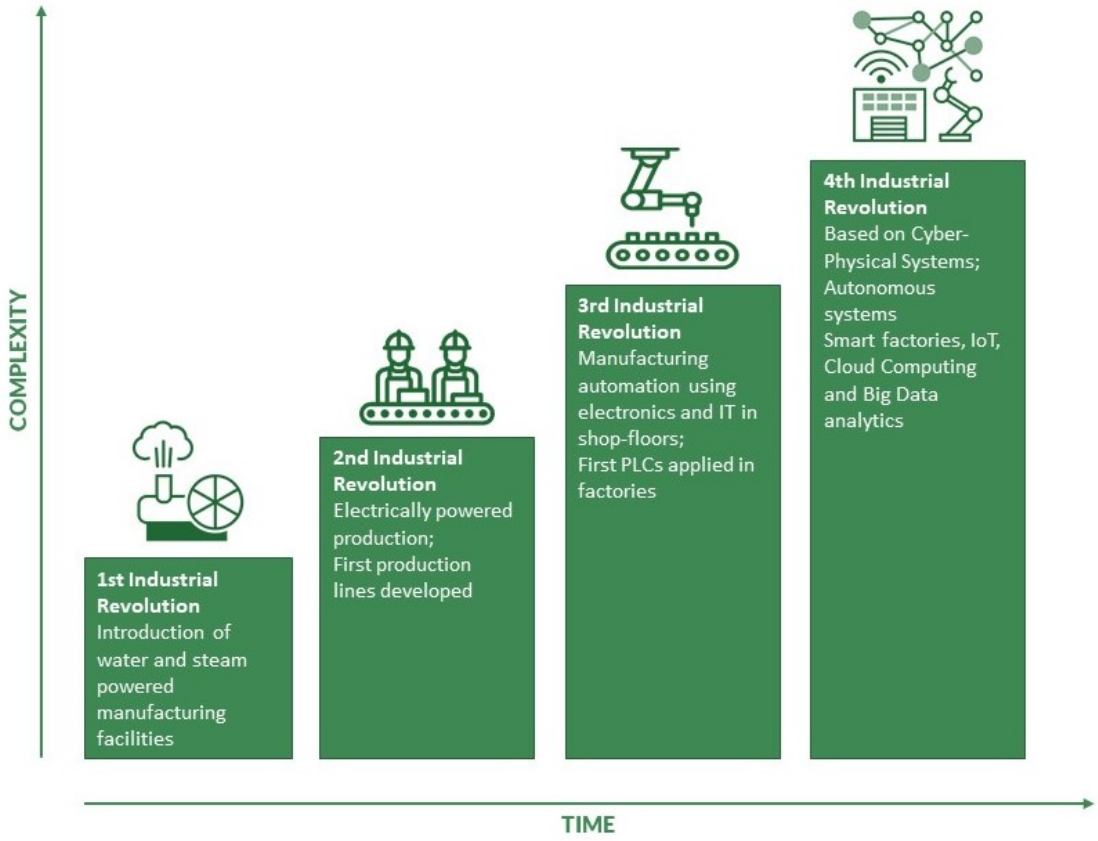

Figura 2.1: Etapas da revolução industrial. Adaptado de [5]

Tal como a introdução da máquina a vapor de James Watt despoletou o início da primeira revolução industrial, as linhas de produção de Henry Ford e a energia elétrica deram o início à segunda e a automação e aplicação de computadores no *shop floor* (PLC) nos levaram à terceira, o movimento da Indústria 4.0 vai ganhando força com a *Internet*, que facilita a criação de redes de comunicação entre máquinas, processos e humanos [5, 6]. A *Internet* será o símbolo da quarta revolução industrial, que permite a implementação das várias novas tecnologias supracitadas. Na verdade, a Indústria 4.0 tem a sua base assente na ideia de acoplar todos os sensores, *software*, dispositivos e todas as ligações de rede para coleção e tratamento de dados através da *Internet* [7, 8]. Este é o conceito IoT, que será abordado adiante.

É em 2011 na *Hannover Messe*, a principal feira mundial para a tecnologia industrial, que se apresenta pela primeira vez o conceito de Indústria 4.0 [9]. Na verdade, rapidamente se tornou a estratégia nacional. Sendo a Alemanha uma das mais competitivas indústrias transformadoras, há um constante esforço para estar à frente da curva face à competitividade crescente e aos baixos custos de produção nos países produtores emergentes. A partir daqui, este termo expandiu-se pelo mundo e surgiram novos conceitos como as *Smart Factories* nos Estados Unidos da América [10] e o plano *Made In China 2025* na China [11].

2.1.2 Princípios de *design*

Como ajuda à implementação de uma visão orientada à Indústria 4.0 foram sugeridos 6 princípios de *design*: interoperabilidade, virtualização, descentralização, capacidade *real-time*, assistência técnica e orientação ao serviço e modularidade [12].

A interoperabilidade refere-se à criação de redes que fazem a ligação entre pessoas, processos, máquinas, entre outros. Está profundamente relacionado com IoT e IoS que abordam a conexão entre todos os participantes numa fábrica, sejam ativos, humanos ou informação e fluxos de dados [12]. Virtualização representa a informação de sensores, modelos de simulação, modelos da planta da fábrica e ainda DTs que possam existir [6, 12]. Deve existir troca de informação entre o domínio físico e virtual para se realizarem previsões e procurar a otimização de processos. Descentralização é o termo que descreve a automatização de tomadas de decisão. O sistema, que deve conter o planeamento de produção, faz o controlo e toma as decisões em tempo real [6, 12]. Contudo, o poder de decisão humano não deve ser ignorado. Com a monitorização *real-time* do estado e do comportamento da produção consegue-se criar uma base para a tomada de decisão em virtude da informação e dos dados recolhidos de forma contínua [12]. Este aspeto está diretamente ligado aos princípios de orientação ao serviço, que representa o desenvolvimento de novos serviços com base nos dados recolhidos, e à modularidade que retrata a flexibilidade e agilidade do sistema reagir às necessidades do cliente com o crescente detalhismo e procura por personalização [12].

2.2 Sistema Ciber-Físico (CPS)

2.2.1 Evolução do conceito

Desde a sua origem, o termo CPS vem-se enquadrando em diversas áreas de estudo, não existindo uma definição clara para este conceito [13]. Não obstante, de acordo com Rajkumar et al. [14], os CPSs são geralmente descritos como sistemas físicos de engenharia que, recorrendo a um núcleo de computação e comunicação, monitorizam, controlam e coordenam operações. Através da *internet*, estes fornecem serviços de acesso e processamento de dados [15], estando por isso intensamente ligados à IoT. Este conceito emergente revela-se uma tecnologia promissora na área da automação industrial e na implementação de *Digital Twins*.

Os CPSs terão sido introduzidos em 2006 num *workshop* realizado pela NSF. Surgem com o intuito de impulsionar a exploração de novas tecnologias que acelerem e tornem mais seguro o desenvolvimento e a integração de sistemas físicos de engenharia, capazes de apresentar um bom desempenho na comunicação, no controlo e na computação [15]. Os principais campos de aplicação de CPSs são, entre outros, a indústria transformadora, e os setores energético, dos transportes e da saúde.

2.2.2 Sistemas Ciber-Físicos e *Digital Twins*

Como referido em 2.1.1, os CPSs estão inseridos nas tecnologias constituintes da indústria 4.0. São um termo que encaixa na visão implementada pela indústria 4.0 e que se relaciona com

as restantes tecnologias associadas, como é exemplo a IoT. O DT, por sua vez, faz também parte do leque de conceitos associados a estes sistemas.

Lee et al. [4] afirmam que perante a necessidade de gerir e analisar dados de processos e de maquinaria, desenvolveram um modelo ciber-físico que garante uma maior eficiência no tratamento de informação. Este modelo é, na verdade, um DT da máquina que na plataforma em nuvem simula o estado da mesma, através dos seus dados físicos e de algoritmos . De uma forma mais simples, num CPS o modelo físico envia os seus dados ao modelo digital, sendo que este faz a monitorização ou atua de volta sobre o domínio físico.

2.3 Internet das Coisas (IoT)

2.3.1 Evolução do conceito

Um aspeto crucial à Indústria 4.0 é a conectividade e interação entre os elementos participantes na produção. Como previamente referido, a IoT é um dos pilares da Indústria 4.0 e promove a conectividade para análise e tratamento de dados [8].

Este termo terá sido pela primeira vez utilizado em 1999 por Kevin Ashton numa apresentação na *Procter & Gamble* (P&G) [16], embora as suas tecnologias fundamentais venham a ser desenvolvidas desde muito antes, como a própria *Internet* que começou com o projeto ARPANET em 1969, ou os sistemas informáticos incorporados que surgem em 1974 [17]. Entretanto vêm emergindo tecnologias que permitem a expansão do conceito IoT, muitas destas aliadas à ideia proposta por Mark Weiser da computação presente mas quase invisível [18]: os nó sensores ou *mote*, as comunicações *wireless* e a eletrónica digital [17].

> "The IoT creates an intelligent, invisible network fabric that can be sensed, controlled and programmed. IoT-enabled products employ embedded technology that allows them to communicate, directly or indirectly, with each other or the Internet." Jim Chase (2013) [19]

Apesar de todo o trabalho desenvolvido no âmbito da IoT, não existe ainda uma definição consensual entre todos os autores [17]. Com o constante introduzir de novas tecnologias o termo vem-se alargando e a sua definição vê-se limitada, sendo em grande parte das vezes orientada ao seu campo de aplicação [19, 17]. Contudo, as suas características fazem uma boa descrição do seu conceito fundamental, tal como ilustrado na Figura 2.2.

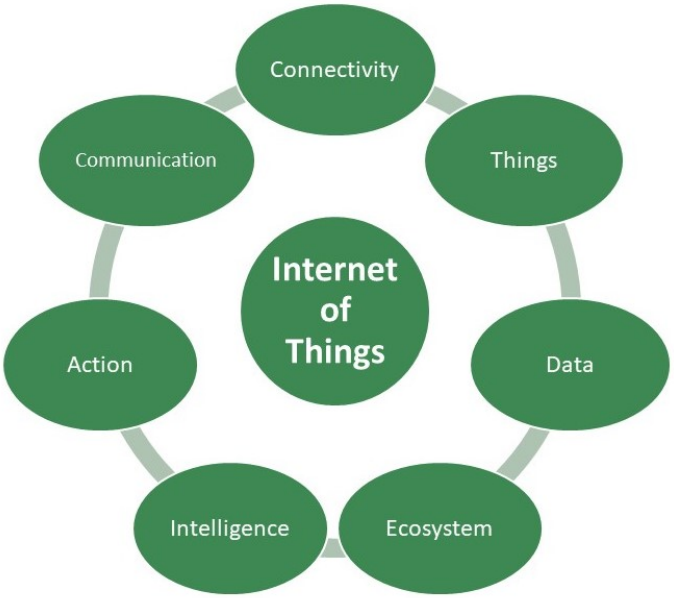

Figura 2.2: Características do conceito IoT. Adaptado de [20]

2.3.2 Objetos inteligentes

Como visto na secção anterior, o termo IoT é introduzido numa apresentação onde Kevin Ashton endereça a ligação da tecnologia RFID, aplicada no *supply chain* da P&G, à *Internet*. Nessa apresentação, Ashton [16, 17] propõe a ideia de coleção e tratamento de dados feita por computadores sem qualquer intervenção humana, com recurso a sensores e à tecnologia RFID.

Esta idealização vem sendo desenvolvida e aperfeiçoada com a ajuda dos objetos inteligentes (*smart objects*). Kopetz [21] define o objeto inteligente como a "coisa" física que ascende com a integração de um dispositivo eletrónico para fornecer inteligência local e ligação ao sistema ciber-físico criado, com ajuda da *internet*, considerando-o o bloco de construção da IoT. De acordo com o trabalho de Miorandi et al. [22], os *smart objects* podem ser definidos como entidades que:

- têm uma vertente física e um conjunto de propriedades físicas associadas;
- têm um conjunto mínimo de funcionalidades de comunicação, como a capacidade de ser localizado, aceitar e responder a mensagens;
- possuem um identificador único;
- são associados a pelo menos um nome (descrição do objeto inteligível a um humano) ou endereço (*string* inteligível a uma máquina);
- possuem capacidades de computação básicas para tomar decisões sobre si próprios ou interagir com entidades externas;
- podem possuir meios de desencadear ações com efeito no domínio físico.

De facto, as "coisas" a que se refere a IoT são geralmente classificadas de objetos, seja uma máquina, um dispositivo, um computador ou um mesmo um objeto virtual [23, 17]. A conectividade que os torna inteligentes vem criando ainda mais conceitos relacionados ao objeto inteligente, como as partes inteligentes e os produtos inteligentes.

Em síntese, a criação de um DT depende grandemente das capacidades que surgem do conceito IoT. Com a informação fornecida pelas tecnologias da IoT é possível espelhar o estado e o desempenho do sistema no DT. Na prática, as ferramentas da IoT vieram tornar o DT uma solução economicamente eficiente e expandir a sua definição e área de aplicação [24].

2.4 *Digital Twin*

2.4.1 Evolução do conceito

A primeira definição de DT no ambiente industrial é introduzida por Michael Grieves, professor no *Florida Institute of Technology*, em 2002 [25]. Surge numa palestra na Universidade de Michigan num contexto de gestão do ciclo de vida do produto, mas com pontos de partida para as características fundamentais ao conceito do DT de hoje - os domínios reais e virtuais, os subespaços virtuais e um fluxo bidirecional de informação e de dados entre o sistema real e o sistema virtual, de acordo com a Figura 2.3.

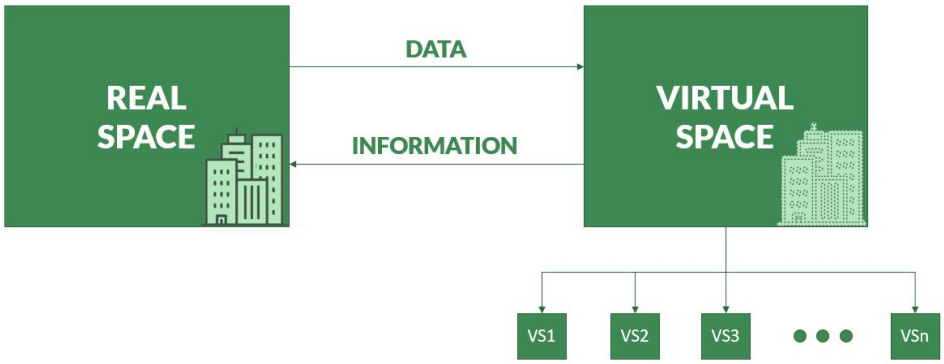

Figura 2.3: Base conceptual do *Digital Twin*. Adaptado de [25]

Neste cenário é implementado um espaço virtual, que corresponde à representação virtual de um produto físico em permanente atualização com o estado do mesmo. É efetivamente criado um "gémeo digital"que espelha o sistema real com os dados que recebe. O DT é então apresentado na sua génese como uma cópia de um produto existente, mas vem-se estendendo aos processos mais complexos que se encontram nas diversas indústrias. Um DT hoje pode representar toda uma fábrica e os seus processos.

Em maio de 2010 a NASA apresenta uma nova definição para o DT, orientada ao ambiente aeroespacial. Neste contexto, o DT é descrito como uma simulação integrada de um veículo ou sistema e que deve dispor, entre outros, dos melhores modelos físicos e atualização de sensores para replicar de forma eficaz o comportamento do seu gémeo físico no espaço [26, 8]. Desde então, como referido por Negri et al. [8], este conceito de DT fundamentado pela NASA vem evoluindo e integrando grande parte dos trabalhos no campo aeroespacial.

À medida que o conceito do *Digital Twin* vai crescendo na indústria aeroespacial, dá-se em paralelo a introdução deste na indústria transformadora com o trabalho de Lee et al. em 2013 [4], que leva o DT para além da sua definição inicial de uma cópia virtual de um produto, descrevendo-o como a representação virtual dos recursos de produção. Nesse mesmo trabalho, o DT de uma máquina real trata-se de um modelo ciber-físico que simula o estado da máquina com o conhecimento proveniente do sistema físico e com algoritmos analíticos. Na verdade, abriram-se as portas para o debate sobre a verdadeira função do DT na indústria transformadora, em particular relacionados com os conceitos da Indústria 4.0, *Big Data* e plataformas *Cloud* [8].

2.4.2 Nível de integração

Como documentado na secção anterior, o conceito DT vem-se transformando e complexificando ao longo do tempo. Com efeito, de acordo com Gichane et al. [27], este terá evoluído de modelos menos desenvolvidos, hoje conhecidos como *Digital Model* e *Digital Shadow*, que diferem apenas na interface entre o produto ou objeto físico e digital, isto é, no nível de integração. Kritzinger et al. [28] sugerem, neste sentido, definições para estes modelos:

- ***Digital Model***

O *Digital Model* caracteriza-se sobretudo pela inexistência de fluxos automáticos de informação. Deste modo, alterações tanto no domínio físico como no domínio virtual não têm qualquer influência um no outro. As trocas de informação são feitas manualmente, representando, por norma, modelos matemáticos de produtos e modelos de simulação de fábricas planeadas [28].

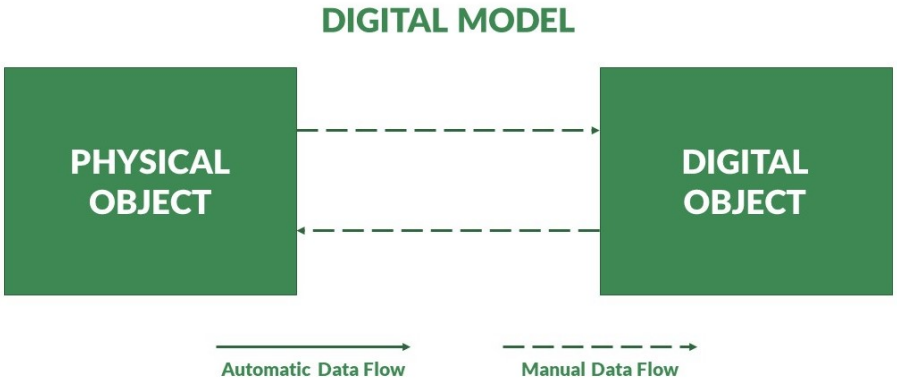

Figura 2.4: Fluxo de dados num *Digital Model*. Adaptado de [28]

- ***Digital Shadow***

O *Digital Shadow* constitui uma melhoria do *Digital Model* uma vez que o "objeto digital"vai sendo atualizado automaticamente com as variações do estado do "objeto físico" [28] com auxílio de, por exemplo, sensores. No entanto, no sentido contrário não se verifica qualquer troca automática de informação. Este padrão vê grande utilização para análise em tempo real e simulação.

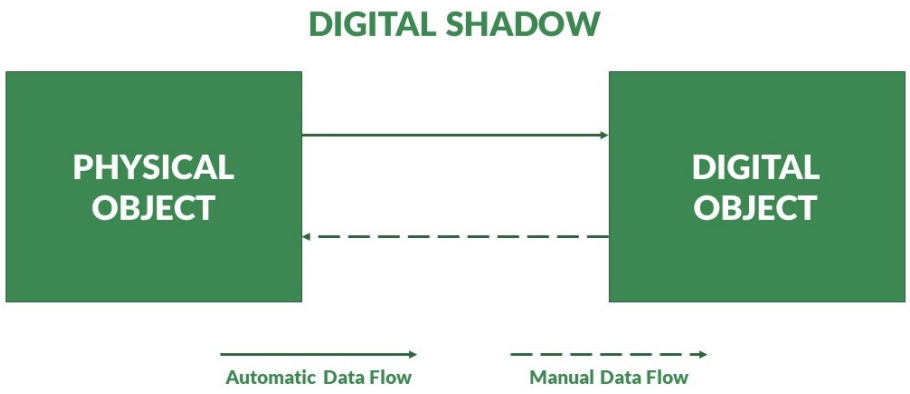

Figura 2.5: Fluxo de dados num *Digital Shadow*. Adaptado de [28]

- ***Digital Twin***

O *Digital Twin*, por sua vez, representa o culminar da automatização das interfaces. Nesta versão, o "objeto digital" poderá funcionar como um controlador [28], dado que as trocas de informação se dão automaticamente nos dois sentidos.

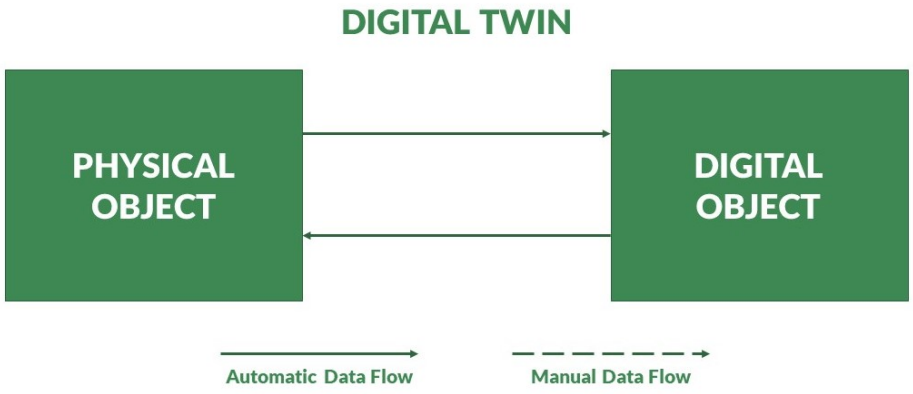

Figura 2.6: Fluxo de dados num *Digital Twin*. Adaptado de [28]

2.4.3 Classificação e aplicações de *Digital Twins*

Não existe um método de classificação bem definido para os DTs, no entanto a flexibilidade destes permite inseri-los em várias áreas industriais. Nos primeiros trabalhos publicados com DTs, procurava-se principalmente criar uma cópia virtual de produtos para análise e comparação de resultados. Mais tarde, com a NASA a incorporar os DTs nos seus *roadmaps* como espelho do veículo em voo [26], esta tecnologia vê a sua aplicação ganhar volume no setor aeroespacial. Porém, nos últimos anos os DTs interessam cada vez mais à indústria transformadora.

O DT de hoje, no que toca à indústria transformadora, é em geral definido como uma representação virtual de partes físicas ou componentes, produtos, processos ou até de sistemas complexos. Insere-se em áreas desde o planeamento e controlo da produção, passando pela manutenção preventiva até ao planeamento de *layouts* de uma fábrica [28].

2.4.4 *Digital Twins* aplicados à produção

No âmbito do planeamento e controlo da produção, Rosen et al. [29] dão o exemplo de um CPS de produção cujas unidades de produção têm um DT a si associados, sendo estas uma estação de cargas e descargas com robô, duas máquinas CNC e um sistema de transporte. Os DTs destas unidades de produção contêm informação como o identificador único, capacidades da unidade, configuração, estado e dados da palete que se encontra a processar. Por outro lado, as partes possibilitam este rastreio dado que também sustentam um DT. Efetivamente, estes DTs contêm informação desde o identificador e tipo de parte ao número de ordem de produção, prioridade, estado, localização e histórico de produção. Estas soluções surgem como resposta a oportunidades de produção levantadas pelo autor no contexto da indústria 4.0, dos CPSs e dos DTs, sendo estas [28, 29]:

- planeamento de ordens de produção com base em dados estatísticos;
- diagnóstico detalhado para melhoramento de apoio à decisão;
- planeamento e execução de ordens de produção de forma automática pelas unidades de produção.

2.4.5 *Digital Twins* de processos

O DT de processos, que é o que interessa a esta dissertação, procura replicar em tempo real todo um processo de fabricação desde o *design* de um produto à sua produção. Neste sentido, o poder do DT desloca-se da visualização e da colaboração de uma peça, por exemplo, para toda uma operação de transformação que nos fornece um maior conhecimento para a tomada de decisões.

De facto, Vachálek et al. [30] desenvolveram um DT de uma linha de produção de pistões hidráulicos. Neste projeto, o modelo digital foi criado com a ferramenta de simulação *Plant Simulation* da Siemens, onde todos os processos associados a esta linha de produção foram detalhadamente mapeados para que o DT pudesse recolher e interpretar a informação de forma precisa. Este

DT visa sobretudo otimizar a produção em função de planos de produção. Efetivamente, a maior diferença que se encontrou entre o tempo de produção do sistema real e do sistema simulado foi de apenas 1 segundo. Em conclusão, os autores deixaram claro que o conceito desenvolvido foi capaz de demonstrar a interação entre os processos de produção reais com o modelo de simulação digital e acrescentam que esta interação pode trazer uma melhor compreensão às dinâmicas dos processos produtivos.

2.4.6 Arquiteturas de sistemas com *Digital Twins*

No caso de ser implementado um DT integrado com simulação, o processo de *twinning* - criação de um DT - possibilitado pela comunicação e sincronização contínua entre o gémeo físico, o gémeo digital e o ambiente externo em que se inserem, consegue fazer a previsão de possíveis defeitos e falhas e permite o teste de diferentes configurações no processo de produção [31]. Este é um dos motivos pelos quais o DT integrado com simulação é extremamente benéfico para monitorização e manutenção preventiva. Para a implementação destes modelos de simulação, é necessário desenhar uma arquitetura que sustente o funcionamento de um DT. Geralmente, as arquiteturas de um DT são compostas pela natureza física e digital do que pretendemos monitorizar ou controlar e por um intermediário que estabelece a comunicação entre ambos.

Assim, Haag e Anderl [32] desenvolveram como prova de conceito uma *test bed* em que consideram o DT uma representação digital de um produto. Neste trabalho, o DT consiste numa representação CAD e é composto pelas propriedades e condições do objeto físico, simulando o seu comportamento durante o seu ciclo de vida. A *test bed* é composta pelo objeto, pelo seu DT e pela interface de comunicação entre ambos. Para a comunicação entre os gémeos foi adotado um padrão *publish-subscribe* baseada no protocolo MQTT. A arquitetura utilizada permite distribuir informação por clientes que subscrevam aos tópicos, contribuindo para a fácil incorporação de mais clientes. Os resultados eram apresentados num *dashboard* em qualquer *browser* com acesso à *internet*. Na Figura 2.7 é apresentado um esquema da arquitetura proposta pelos autores [32].

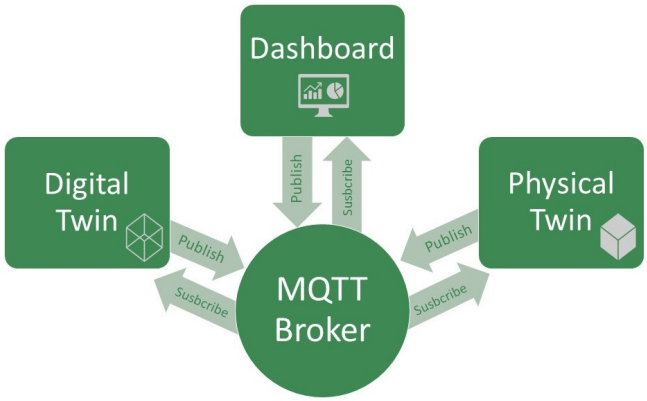

Figura 2.7: *Overview* e arquitetura do DT. Adaptado de [32]

2.5 Criação de modelos de Simulação de forma automática

Um DT pode ser descrito como um modelo vivo e inteligente que visa monitorizar e controlar os processos do seu gémeo físico ao seguir o seu ciclo de vida [31]. Por outro lado, os modelos de simulação são frequentemente tomados como *Digital Twins*. Face a este equívoco, Shao et al. [33] afirmam que um DT pode ser um modelo de simulação mas o contrário pode não se verificar. Acrescentam que isto se deve ao facto de os *Digital Models* usados em simulação, descritos na secção 2.4.2, conterem o mesmo tipo de informação sensorial que um DT, mas cuja informação poderá ter sido gerada ou manipulada [33]. Em suma, a simulação poderá não estar a replicar o que está a acontecer em *real-time*, mas o que poderia acontecer com as manipulações aplicadas por um utilizador ou algoritmo.

Um tema que pode ser interessante como complemento ao projeto principal e associado à simulação é a geração automática de modelos de simulação. No sentido do trabalho a desenvolver, permitiria a criação do DT a partir de um ficheiro com a configuração do *shop floor*, colocando todos os blocos constituintes com os respetivos dados no motor de simulação.

Son et al. [34] apresentaram uma arquitetura para geração automática de modelos de simulação. Esta foi desenvolvida para modelos detalhados aplicados ao controlo do *shop floor* baseado em simulações em tempo real. Este modelo de simulação é criado a partir de dois modelos do *shop floor* - um de recursos e outro de controlo - que fornecem as informações a si associadas. Deste modo, o código de simulação gerado pode ser utilizado no controlo do sistema de produção através do envio e receção de mensagens com recurso a uma ligação *Ethernet* ao sistema de execução de alto nível (MES). A arquitetura desenvolvida pelos autores é apresentada na Figura 2.8.

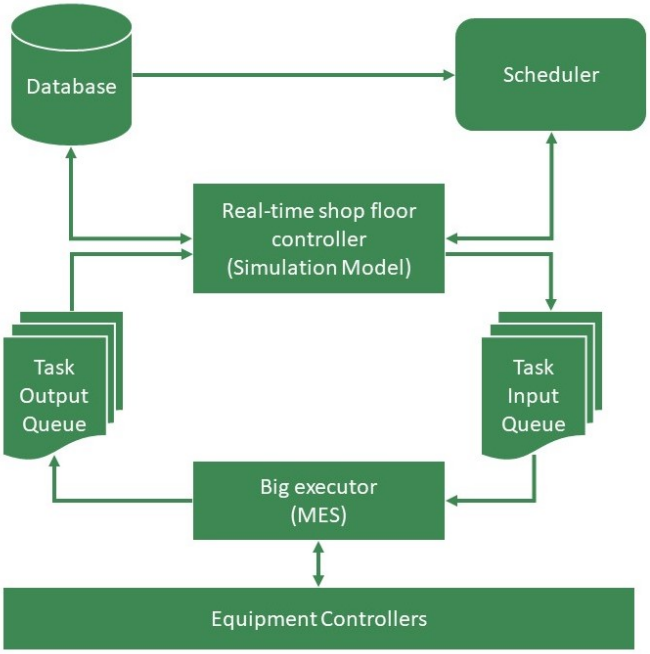

Figura 2.8: Arquitetura para controlo do *shop floor* baseado em simulação. Adaptado de [34]

Esta estrutura é composta por uma base de dados (*Database*) que acompanha os pedidos de peças, um planeador (*Scheduler*) capaz de realizar atividades de agendamento a partir do modelo de simulação, filas de entrada e saída de tarefas que fazem a comunicação e divisão entre o módulo de execução e o tomador de decisão, um modelo de simulação e um MES (apelidado de *Big executor* pelos autores) responsável pela execução de tarefas e por acompanhar o estado do equipamento.

2.6 Conclusão

O levantamento dos fundamentos teóricos contribuirá certamente para a análise e desenvolvimento da solução para o problema que será abordado no Capítulo 3. De facto, o estudo dos conceitos associados ao DT - Indústria 4.0, CPS e IoT - proporciona uma visão mais ampla da natureza do DT, nomeadamente o porquê, como e quando recorrer à implementação deste. Esta revisão bibliográfica será ainda um excelente apoio ao trabalho a desenvolver, dado que apresenta algumas arquiteturas de sistemas com DT previamente criadas e utilizadas por autores devidamente referenciados. De referir que, por razões de planeamento do projeto, o tópico de criação de modelos de simulação de forma automática não foi considerada.

Em síntese, todos os conteúdos estudados ao longo deste capítulo têm a capacidade de enriquecer a solução a desenvolver, sendo esta análise um benefício à potencial aplicação destes temas no projeto, ou uma referência para aplicação num trabalho futuro.

Capítulo 3

Problema e Metodologia

O presente capítulo aborda de forma sistemática a criação de uma *framework* para um DT aplicado a um *shop floor*, o problema em estudo. Neste sentido, serão aprofundados três componentes distintas cruciais ao desenvolvimento desta estrutura - monitorização em tempo real, modo de simulação de eventos futuros e otimização da produção.

Com a crescente competitividade na indústria transformadora, as empresas procuram cada vez mais estar à frente da curva com o impulso das mais recentes inovações tecnológicas. Os DT revelam-se uma das mais promissoras tecnologias neste setor graças às vantagens que a sua implementação pode trazer no domínio da visualização e monitorização do estado de uma fábrica.

Adicionalmente, harmonizar as capacidades de um DT com a simulação de eventos discretos e com a habilidade do DT comunicar com aplicações inteligentes irá certamente integrar uma plataforma única e distinta do DT comum. Por norma, este último é planeado num contexto exclusivo sem a versatilidade para se adaptar a diferentes tipos de produção. Quer isto dizer que surge o interesse na criação de um DT (e respetiva *framework*) capaz de se ajustar aos diferentes *layouts* do *shop floor*.

3.1 Metodologia desenvolvida

Efetivamente, os *digital twins* são de forma geral produzidos em cenários específicos e de acordo com as necessidades das operações do *shop floor* em questão, como referido na secção anterior. A plataforma que se pretende desenvolver, no entanto, procura garantir adaptabilidade a diferentes tipos de processos de manufatura e tem também como objetivo atingir a interoperabilidade com serviços inteligentes externos que podem ser aplicados sobre a operação do *shop floor*.

Deste modo, o sistema a desenvolver foi dividido em 3 componentes distintas, apresentadas na Figura 3.1. Estas componentes são a monitorização em tempo real, característica fundamental e que constitui em grande parte a criação do DT, o modo de simulação com base em planos de produção do utilizador e a comunicação com serviços externos, que neste caso se trata de uma ferramenta de otimização de planos de produção. A primeira destas três componentes contribui

para a conceção do DT, sendo que as restantes suportam a estrutura do sistema a desenvolver, enriquecendo o primeiro com novas funcionalidades.

Como prova da flexibilidade da *framework*, as diferentes componentes foram aplicadas a diferentes tipos de processos. Enquanto que a monitorização em tempo real e a simulação foram testadas em processos do tipo *job shop*, a otimização foi aplicada a processos do tipo *flow shop* seguindo um *layout* em linha. De facto, as três componentes estão preparadas para serem empregues em qualquer tipo de processos, sendo esta adaptabilidade uma característica que acrescenta valor à *framework*.

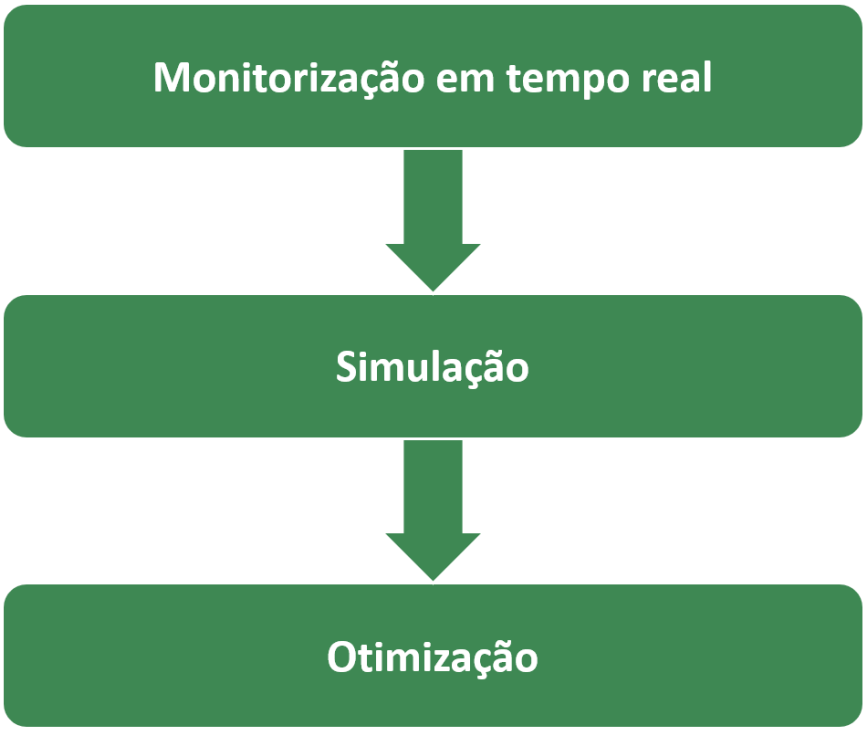

Figura 3.1: Componentes do desenvolvimento da *framework*

Num primeiro momento, a monitorização em tempo real viabiliza a representação gráfica e supervisão do *shop floor* a qualquer momento. Esta componente trata os dados que são enviados pela fábrica, traduzindo-os para o modelo 3D e replicando de forma fiel o que está a acontecer a cada instante na área de trabalho. O modo de simulação, por seu turno, explora o potencial do motor de simulação para fazer simulações com planos de produção definidos pelo utilizador. Por fim, a etapa de otimização visa demonstrar a capacidade da plataforma interoperar com ferramentas externas, sendo que neste caso se recorreu a um programa de otimização de planos de produção. A Figura 3.2 apresenta o diagrama de casos de uso do sistema.

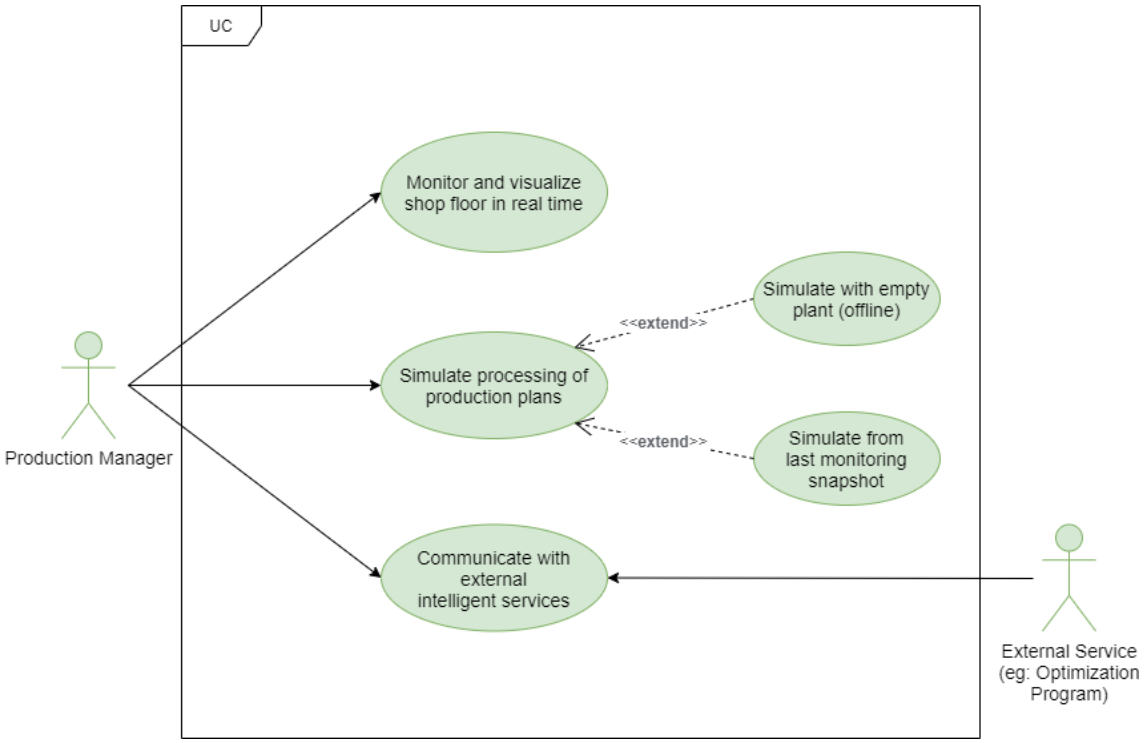

Figura 3.2: Diagrama de casos de uso

3.2 Arquitetura proposta

A abordagem considerada passa pela criação de uma *framework* que faz a interação entre os sensores e atuadores no *shop floor* e a cópia digital desenvolvida com um motor de simulação. Para a proposta de solução do problema em questão, considerou-se o DT a cópia virtual do sistema físico.

O emulador do *shop floor* será desenvolvido a partir de um ficheiro CSV que é um *log*, ou registo, de todos os eventos relevantes à monitorização que ocorreram na fábrica. Num caso real, este ficheiro corresponderia às mudanças de estado nos sensores e atuadores presentes no *shop floor*. Por conseguinte, o DT deve comunicar com este sistema para receção e interpretação dos dados. Esta troca de mensagens é feita com base num padrão *publish–subscribe*, onde o sistema real publica a sua informação e o sistema digital os recebe pela subscrição desse chamado "tópico". Esta ponte é feita com recurso ao Apache Kafka, *software* de processamento de fluxos de dados que oferece uma alta taxa de transferência a uma latência baixa para um processamento mais rápido, sendo bastante importante ao processamento de dados em tempo real. A representação virtual, gerada com recurso à ferramenta de *software FlexSim*, é atualizada pela aplicação que recebe e faz o tratamento de dados recebidos - desenvolvida em *Python* - através de *sockets*. Esta aplicação é também responsável pela atualização da base de dados, desenvolvida em *MongoDB*. A arquitetura proposta é ilustrada na Figura 3.3.

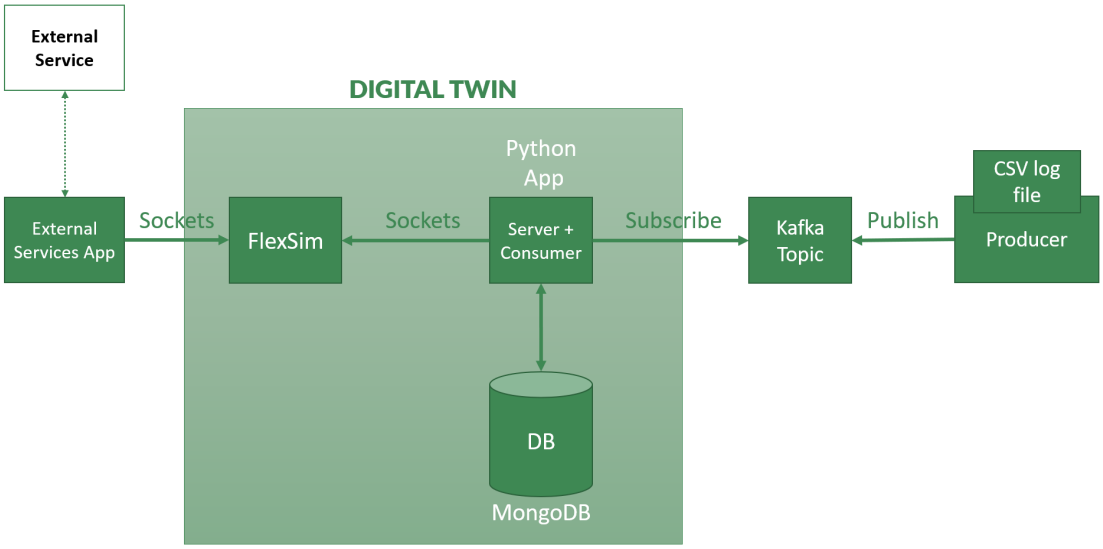

Figura 3.3: Arquitetura proposta

No *FlexSim* deve ser possível fazer a paragem da monitorização em tempo real feita pelo DT, quase como que cortando a sua comunicação com o sistema físico, para fazer a simulação de possíveis cenários futuros com configurações diferentes do *shop floor*.

Por fim, a comunicação com serviços externos inteligentes deve também ser promovida. A plataforma deve ser capaz de comunicar com serviços fora da máquina local que possam impactar diretamente sobre o sistema. Neste caso em particular, recorre-se a um programa de otimização de planos de produção.

Capítulo 4

Monitorização em Tempo Real

Neste capítulo é descrita a estrutura de implementação da componente de monitorização em tempo real do *shop floor*. A monitorização tem por base uma aplicação *Python*, que atuará como servidor, responsável por fazer o tratamento dos dados que são enviados pela fábrica e por estabelecer a comunicação com a ferramenta de *software FlexSim* para que esta última receba a informação necessária filtrada. Para além disso, esta aplicação está também encarregue atualizar a base de dados criada em *MongoDB* com cada evento que ocorre na fábrica. Este paralelismo é cumprido com recurso a *multithreading*, que possibilita a execução de vários segmentos de código - ou *threads* - em simultâneo. O *FlexSim*, por sua vez, constitui a visualização 3D do *shop floor* e implementa a lógica de circulação das peças pelas estações de trabalho (*workstations* ou WS) com base na informação enviada pelo servidor, através de um fluxograma de processos. Este modelo 3D é essencialmente formado pelas peças que vão entrando na fábrica e por estações de trabalho, sendo cada uma precedida de um *buffer* de entrada. Ao longo do presente capítulo, todos estes conceitos serão debatidos de forma detalhada. A estrutura desta componente é ilustrada na Figura 4.1.

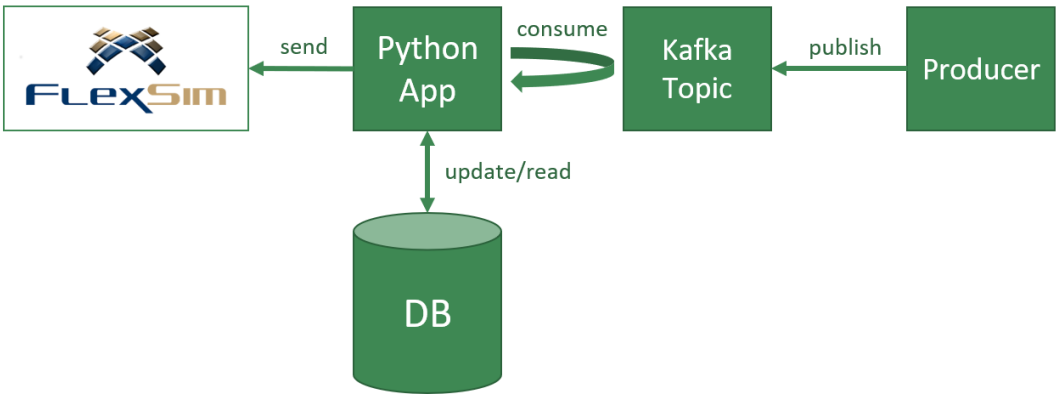

Figura 4.1: Estrutura da componente de monitorização em tempo real

4.1 Servidor *Python*

4.1.1 Comunicação via *sockets*

Para que haja troca de informação entre o servidor (aplicação *Python*) e o cliente (*FlexSim*) é necessário estabelecer um protocolo de comunicação. Feita uma breve análise das funcionalidades de comunicação oferecidas pelo *FlexSim*, optou-se pela comunicação por *sockets* TCP/IP. O protocolo TCP não só garante que os pacotes perdidos na rede são retransmitidos como também faz a leitura dos dados recebidos pela ordem de envio, assegurando que todos os eventos são replicados no modelo 3D pela ordem em que realmente ocorreram.

Do lado do servidor, com auxílio do módulo *socket*, primeiramente é estabelecida a porta em que se vai realizar a comunicação. A *socket* é vinculada ao endereço e fica à escuta de pedidos de conexão, que neste caso serão feitos pelo cliente que é o *FlexSim*.

O *FlexSim*, no que lhe diz respeito, dispõe de *Model Triggers* que permitem executar segmentos de código em contextos específicos. Deste modo, quando se dá início à monitorização - *OnRunStart* - é inicializada a utilização de *sockets*, cria-se a *socket* do cliente com protocolo TCP/IP e é feita a conexão à porta definida no servidor que deve estar à escuta. A partir desse momento, a ligação está estabelecida e pode haver troca de dados. A conexão é apenas fechada caso o servidor ou o cliente sejam encerrados ou caso a monitorização seja parada - *OnRunStop*. Neste último cenário, o servidor continuará à escuta de novos pedidos de ligação até que o servidor seja terminado. As etapas desta comunicação, desde a criação das *sockets* ao seu encerramento, são ilustradas no diagrama da Figura 4.2.

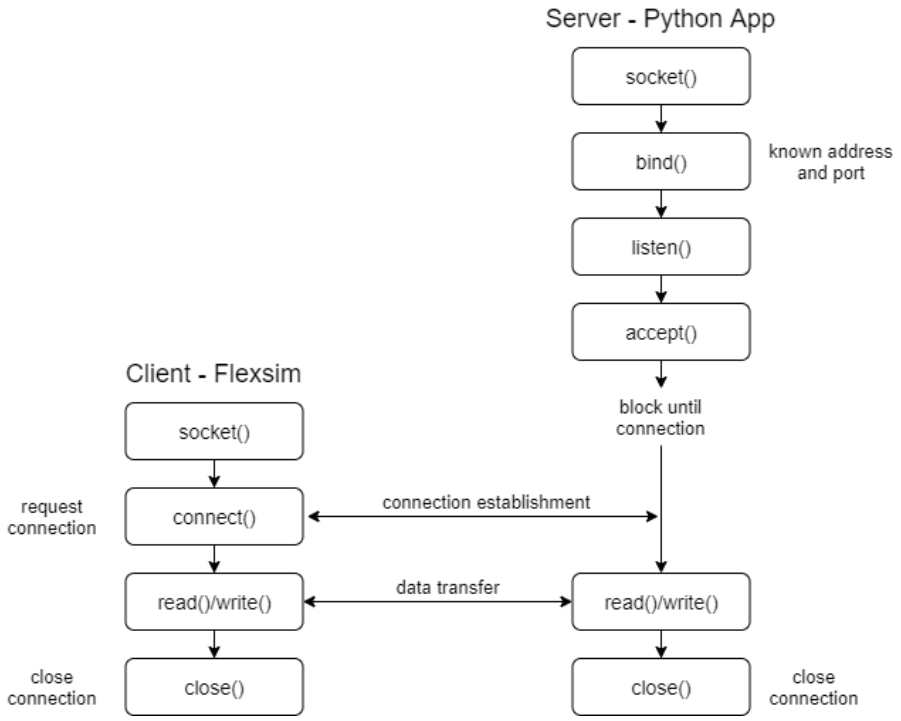

Figura 4.2: Diagrama de comunicação TCP/IP entre o cliente e o servidor

4.1.2 Emulação do comportamento do *shop floor*

Para a emulação do comportamento do *shop floor*, gerou-se um ficheiro CSV com uma sequência de eventos, em que cada linha corresponde a um evento. Este ficheiro, que também podem ser representado em tabelas como é exemplo a Tabela 4.1, traduz o que está a acontecer no sistema físico. Cada evento é composto por quatro parâmetros a seguir listados, que retratam de forma objetiva a movimentação na zona de produção:

- **TIME_OF_OCCURANCE (TOO)** - Determina o instante em que o evento decorre;
- **DESCRIPTION (DESC)** - Descreve o tipo de evento:
  a. ON_ENTER se a peça entra num *buffer* ou WS;
  b. ON_FINISH se o processamento da peça na WS termina;
  c. ON_EXIT se a peça deixa o elemento em que se encontra.
- **ASSOCIATED_PRODUCTION_AREA (APA)** - Identifica o *buffer* ou WS em que se dá o evento;
- **ASSOCIATED_AGENT (AA)** - Identifica a peça associada ao evento.

Por fim, existem algumas restrições que devem ser respeitadas: as peças devem passar pelos *buffers* das respetivas WS antes de entrarem nestas, o identificador de uma peça apenas se pode encontrar uma vez na fábrica em simultâneo e as peças devem deixar uma WS somente após o término do seu processamento.

Tabela 4.1: Excerto de ficheiro CSV utilizado na validação

TOO	DESC	APA	AA
0	ON_ENTER	B7	A1
0	ON_ENTER	B2	A2
0	ON_ENTER	WS7	A1
0	ON_ENTER	WS2	A2
2	ON_FINISH	WS2	A2
2	ON_EXIT	WS2	A2
2	ON_ENTER	B1	A2
2	ON_EXIT	B1	A2
3	ON_FINISH	WS7	A1

4.1.3 Padrão *publish-subscribe*

Como solução para a comunicação entre o servidor (que faz o papel de consumidor de mensagens) e o emulador do *shop floor* (denominado produtor de mensagens), adotou-se um modelo *publish-subscribe* recorrendo à plataforma de processamento de correntes de eventos *Apache*

Kafka. Neste modelo, o produtor de mensagens publica mensagens num tópico criado, sem conhecimento dos subscritores do tópico. Os consumidores de mensagens que estão subscritos ao mesmo tópico, por sua vez, têm acesso a qualquer informação aí publicada. Este padrão de transmissão de mensagens garante uma maior escalabilidade da rede.

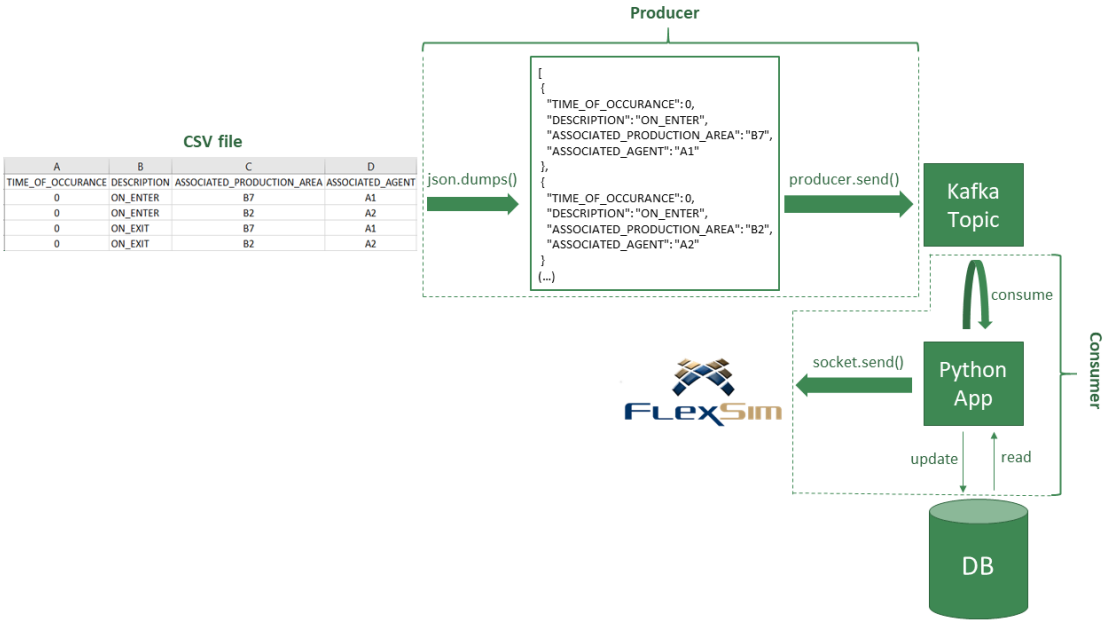

Figura 4.3: Percurso das mensagens segundo o padrão *publish-subscribe*

4.1.3.1 *Publisher* - Produtor

Como visto anteriormente, existe um ficheiro CSV responsável pela emulação da fábrica. Efetivamente, este ficheiro é lido por um programa produtor de mensagens desenvolvido em *Python* e exterior ao servidor, tal como sugere a Figura 4.3. A criação deste produtor surge do módulo *KafkaProducer*, que permite criar um cliente com a capacidade de publicar informação num chamado *Kafka Cluster*.

Posteriormente, o ficheiro que emula o comportamento da fábrica, apresentado em 4.1.2, é lido e cada linha é extraída para uma lista com auxílio do módulo *Python csv*. Construída a lista, é possível criar um dicionário - tipo de estrutura de dados - que facilita a conversão da informação para o formato JSON. Enquanto que um dicionário representa uma forma de trabalhar com informação num programa, JSON é um formato de serialização utilizado para armazenamento e troca dados entre programas, que permite consequentemente fazer *parsing* automático dos dados.

4.1.3.2 *Subscriber* - Consumidor

O consumidor, por seu turno, é criado na aplicação *Python* que também tem o papel de servidor. Este é inicializado com o módulo *KafkaConsumer* numa *thread* dedicada ao consumo de

mensagens e atualização da base de dados. Assim que o consumidor detete a existência de conteúdo no tópico, este vai consumindo as mensagens aí publicadas pela ordem de publicação. De referir que a cada mensagem consumida é feito um *commit* pelo consumidor que atualiza o *offset*, opção que vai marcar a última mensagem processada com sucesso. Assim, caso haja perda de comunicação entre o consumidor e o tópico, o novo consumidor que se conectar tem informação sobre onde continuar o processamento de mensagens através do *offset*.

As mensagens consumidas são então encaminhadas para o *FlexSim*, como veremos em 4.3, e utilizadas para atualizar a base de dados, como descrito em 4.2. A interação do servidor/consumidor com o *Kafka Cluster*, a BD e o *FlexSim* através da *thread* de consumo de mensagens é estruturada no diagrama de sequência da Figura 4.4.

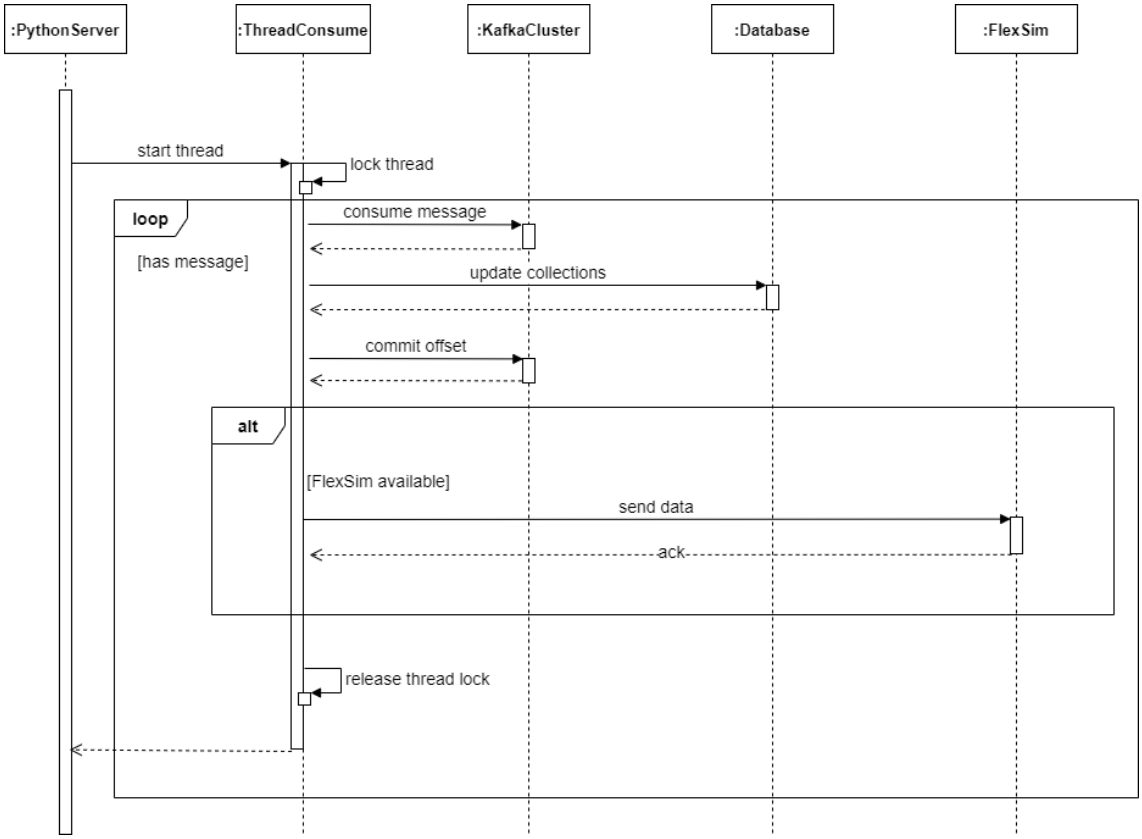

Figura 4.4: Diagrama de sequência do consumo de mensagens

4.2 Base de Dados - *MongoDB*

O papel principal da base de dados (BD) passa por guardar uma *snapshot*, ou fotografia, do último estado do *shop floor*. Deste modo, a cada evento que se dá a BD deve ser primeiramente atualizada. Esta fotografia será consultada sempre que o modelo 3D no *FlexSim* é inicializado, considerando que é possível haver peças em processamento nesse momento. Neste sentido recorreu-se ao *software MongoDB*, programa de base de dados *NoSQL* que guarda informação em documentos

tipo JSON, chamados BSON (*Binary* JSON). Trata-se de uma BD não relacional que se destaca pela sua flexibilidade e escalabilidade, enquadrando-se facilmente nos padrões das tecnologias IoT.

4.2.1 Estrutura da Base de Dados

A BD é composta por duas coleções que guardam informação sobre as estações de trabalho e *buffers*, e sobre as peças em processamento: *objects* e *agents*, respetivamente. O diagrama da BD é apresentado na Figura 4.5.

Figura 4.5: Diagrama da Base de Dados

Os elementos da coleção *objects* são compostos por 3 parâmetros:

- **ID** - *String* com o identificador do *buffer* (B*x*) ou da estação de trabalho (WS*x*);
- **Status** - *String* com o estado do objeto, que pode ser:
    a. *Idle*, caso o objeto se encontre livre para receber peças;
    b. *Processing*, caso a WS esteja a processar uma peça;
    c. *Blocked*, caso uma peça esteja a ocupar a WS;
    d. *Releasing*, caso pelo menos uma peça esteja no *buffer* à espera que a respetiva WS fique livre.
- **Agent** - *Array* com o(s) identificador(es) da(s) peça(s), caso alguma se encontre no objeto.

Os elementos da coleção *agents*, da mesma forma, são formados por 3 parâmetros:

- **ID** - *String* com o identificador da *peça* (A*x*);
- **Status** - *String* com o estado da peça, que pode ser:
    a. *Idle*, caso a peça esteja parada;
    b. *Processing*, caso a peça esteja a ser processada;

- **Location** - *String* com a WS ou *buffer* em que se encontra a peça.

As Figuras 4.6 e 4.7 apresentam, de forma respetiva, excertos das coleções *objects* e *agents* que demonstram o que foi descrito até aqui.

Figura 4.6: Excerto da coleção *objects*

Figura 4.7: Excerto da coleção *agents*

Com esta informação, é possível identificar o que está na fábrica, onde e em que circunstâncias. Exemplificando, as Figuras apresentadas mostram que existem duas peças A1 e A2 na WS3 e WS1, respetivamente. A primeira está parada enquanto que a última se encontra a ser processada.

4.2.2 Comunicação com a aplicação *Python*

Para que a aplicação *Python* comunique com a base de dados, é na primeira que se cria um cliente para uma instância *MongoDB* com auxílio do módulo *PyMongo*, que contém as ferramentas necessárias para a interação com a base de dados. A BD é sempre completamente limpa no arranque da aplicação *Python* a fim de evitar conflitos com o estado atual do *shop floor*, dado que o último registo na BD poder-se-ia encontrar num estado diferente. Após esta inicialização, a BD entra em cena em dois momentos: para ser consultada quando um cliente se liga ao servidor *Python* e para ser atualizada à medida que as mensagens enviadas em tempo real são consumidas, de acordo com a Figura 4.3.

Como referido, a consulta da BD é realizada sempre que um cliente *FlexSim* se liga ao servidor. Neste cenário é percorrida toda a coleção *agents* e, na ocasião de se encontrar um registo, são criadas e enviadas mensagens para o *FlexSim* consoante o número de peças registadas na coleção.

Em suma, a fotografia do *shop floor* guardada pela BD permite ao DT recuperar o estado do processamento para dar início à monitorização visual.

A atualização da BD é recorrente e é feita na *thread* de consumo de mensagens. Trata-se de um aspeto crucial ao bom funcionamento da *framework*, sendo por isso a primeira função a ser executada na aplicação *Python* após a receção de uma mensagem. Logo que a mensagem é recebida, esta é convertida do formato JSON para um dicionário *Python* - o processo inverso do que fez o produtor em 4.1.3.1 - e a partir daí são atualizadas as WS e *buffers* nas coleções da BD.

A atualização é feita conforme o tipo de evento da mensagem. Se a peça entrou num objeto, o estado deste deve ser alterado na coleção *objects* e uma peça deve ser criada na coleção *agents*. Se a peça terminou o seu processamento, o estado do objeto deve ser uma vez mais alterado, bem como o estado da peça. Se a peça saiu de um objeto, o estado do objeto volta a ser atualizado e a peça é removida da coleção *agents*.

4.3 Processamento em tempo real - *FlexSim*

Com o *FlexSim*, pacote de *software* de simulação de eventos discretos, é possível não só ter uma visualização 3D do que ocorre no *shop floor*, mas também simular o comportamento dos sistemas em estudo. Na verdade, esta ferramenta oferece uma biblioteca de recursos que podem ser implementados no modelo, cruciais à criação e implementação do DT. A lógica de movimentação de peças no *shop floor* é feita de forma a maximizar a flexibilidade da *framework*, no âmbito de se concretizar um modelo capaz de lidar com a monitorização em tempo real e com a simulação de cenários futuros em simultâneo. Deste modo, recorreu-se à modelação via fluxograma de processos - *Process Flow* - desenvolvida no *FlexSim* em paralelo com as ferramentas de modelação 3D. A tarefa de otimização será também implementada com recurso ao fluxograma de processos.

4.3.1 Peças

No modelo de simulação, as peças são representadas por caixas, modelo padrão definido pelo *FlexSim*. Estas são criadas em *Sources*, percorrem os *buffers* e estações de trabalho e por fim são retiradas do *shop floor* em *Sinks*. Tal como demonstra a Figura 4.8, cada peça contém informação sobre si própria guardada em *Labels*: o tipo da peça, que habilita a identificação da respetiva rota, o identificador da ordem de produção a que está associada e a linha em que se encontra atualmente na Tabela de Processamento 5.1, tabela responsável pela orientação das peças no modo de simulação que será aprofundado no Capítulo 5.

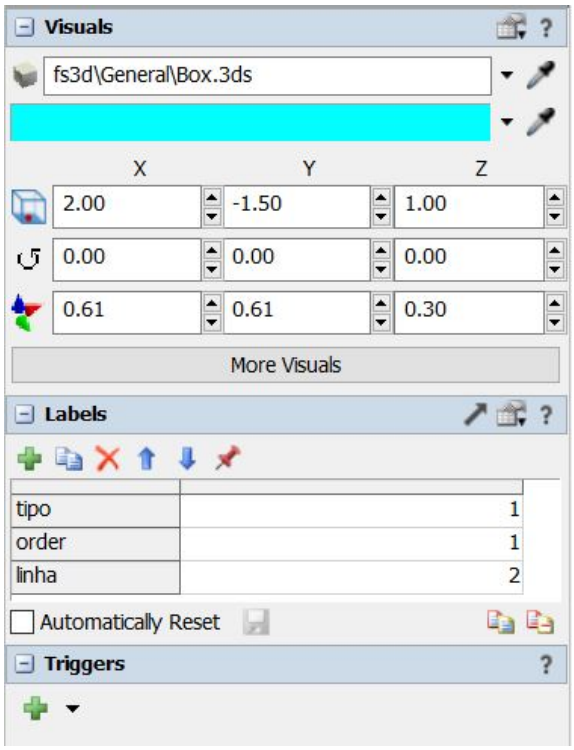

Figura 4.8: Propriedades das peças

As peças são criadas inicialmente numa fila onde são depois redirecionadas para o *buffer* da primeira WS onde vão ser processadas. O processamento é feito pela ordem de chegada e as peças devem aguardar nos *buffers* da WS respetiva até que esta fique livre, dado que apenas uma peça pode ser processada de cada vez numa WS.

4.3.2 Estações de trabalho

As WS são consideradas recursos fixos e denominados *Processors* segundo o *FlexSim*. Como anteriormente referido, precedem um *buffer* de entrada onde se forma a fila de peças a processar. Não apresentam um tempo de processamento fixo, variando com o tipo da peça. A representação gráfica é apresentada na Figura abaixo.

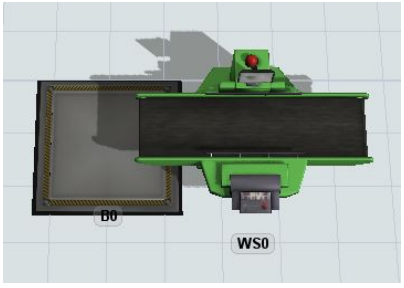

Figura 4.9: Modelo 3D de uma estação de trabalho (à direita) e respetivo *buffer* (à esquerda)

Como referido, as WS têm uma capacidade máxima de 1, enquanto que os *buffers* têm capacidade infinita. Quer isto dizer que o *buffer* apenas liberta uma peça caso a sua WS esteja vazia.

O tempo de processamento em cada WS é ajustável, podendo ser um valor fixo ou um valor que varia com o tipo de peça, desde que este seja indexado numa Tabela de Processamento como são exemplos as Tabelas 5.1 e 5.2, que serão abordadas posteriormente no Capítulo 5.

4.3.3 Fluxograma de processos

O fluxograma de processos da Figura 4.10 é o principal responsável pela movimentação do modelo 3D do *shop floor*, isto é, a criação do *Digital Twin*. Aquando da inicialização deste modelo, é executado o bloco *Source* e é criada uma *token* que percorre o fluxograma de forma descendente, processando as instruções dadas por cada bloco.

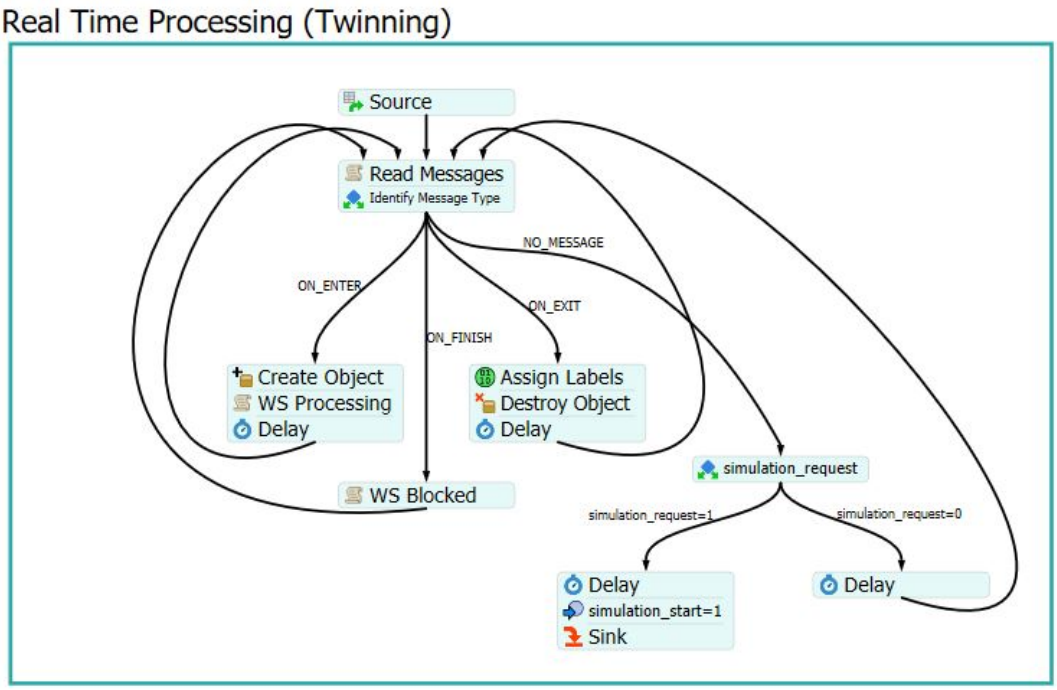

Figura 4.10: Fluxograma do processamento em tempo real

Primeiramente, é recebida e processada num bloco de código *Read Messages* a mensagem enviada pelo servidor. Efetivamente, no caso de ter sido enviada uma mensagem, este trecho de código faz a receção da informação remetida pela *socket* e o *parsing* da mensagem com o método *JSON.parse()* que retorna a *string* no formato *Variant*. Assim, os parâmetros desta mensagem são estruturados num *Map* que permite um fácil acesso e manipulação da informação, como demonstra o esquema da Figura 4.11.

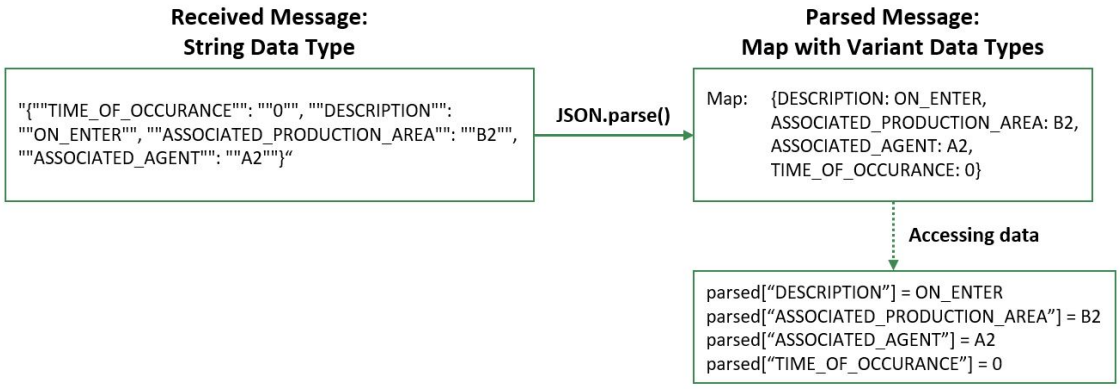

Figura 4.11: *Parsing* da mensagem enviada pelo servidor

É também atribuida uma *label* à respetiva *token*, denominada *jsonMessage*, que vai conter a mesma mensagem pós-*parsing*. A *label* é consultada ao longo do fluxograma de processos e é visível clicando na *token* como mostra a Figura 4.12, onde se encontram também outras propriedades de uma dada *token*. Este bloco termina a sua execução com o envio de uma mensagem de terminação pela *socket*, dando sinal ao servidor para prosseguir. Na condição de não haver uma mensagem inicialmente enviada pelo servidor, a *token* é simplesmente etiquetada com uma *string* vazia na mesma *label jsonMessage*, terminando aí a execução deste bloco.

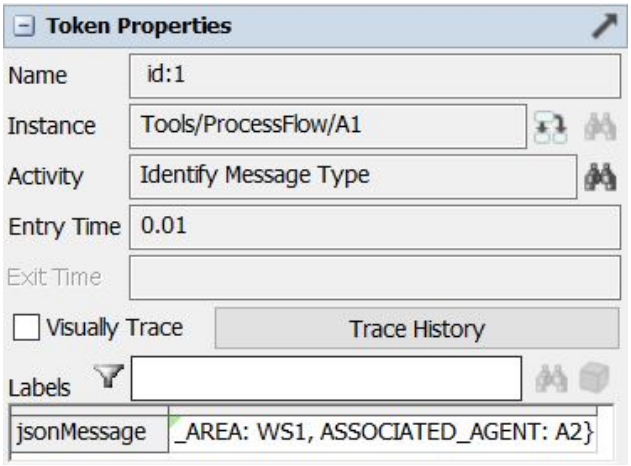

Figura 4.12: Propriedades e *labels* de uma *token*

Feita a análise da mensagem, existem quatro caminhos que a *token* pode seguir dependendo do tipo de evento. Na hipótese da *label jsonMessage* estar vazia, isto é, de não existir uma mensagem recebida, a *token* entra num *loop* em que fica à espera de mensagens verificando sempre os pedidos de simulação, componente aprofundada no Capítulo 5. Se porventura o evento for um dos três abordados em 4.1.2, a *token* deve seguir o respetivo ramo em conformidade com a Figura 4.10, que são:

- ON_ENTER - é criada, no modelo 3D, uma peça na respetiva WS ou *buffer* pelo bloco *Create Object* e é atribuída a *label "tipo"*, que guarda o tipo da peça, à peça criada. Posteriormente, é alterado o estado da WS (se for o caso) associada ao evento para *"Processing"*, indicando que esta está ocupada. Caso se trate de uma entrada num *buffer*, o *FlexSim* trata de alterar o estado do mesmo para *"Releasing"*;
- ON_FINISH - é alterado o estado da WS para *"Blocked"*, indicando que esta não está a processar mas que se encontra ocupada por uma peça que aguarda a sua saída;
- ON_EXIT - é atribuída à *token* a *label "agent"* que deve conter o identificador da peça. Com isto, no bloco *Delete Object*, a *label* mencionada indica a peça ser apagada do modelo 3D. No caso de se tratar de uma saída de uma WS o estado desta é alterado para *"Idle"*, enquanto que no caso de um *buffer* o estado é alterado automaticamente pelo *FlexSim* para *"Empty"*, se aquele ficar vazio.

Terminada a execução dos blocos de cada evento, a *token* volta ao início do fluxograma onde espera por mais mensagens do servidor. O diagrama da Figura 4.13 demonstra de forma resumida e simplificada a lógica tecida para esta componente de visualização 3D do processamento em tempo real, que foi descrita.

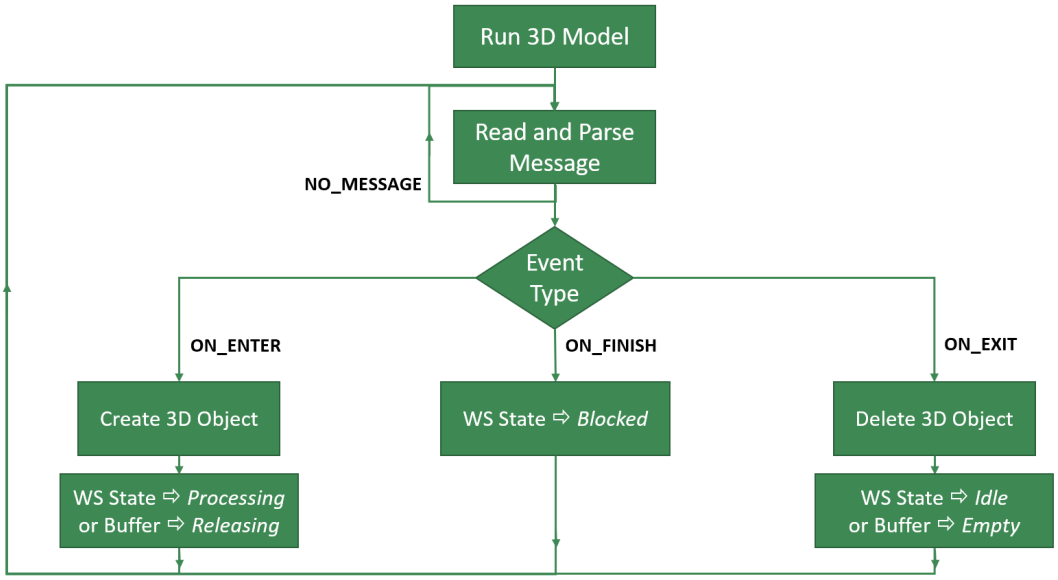

Figura 4.13: Diagrama representativo da lógica do processamento em tempo real no modelo 3D

4.4 Validação e resultados

Para a validação da componente de monitorização em tempo real desenvolvida, considerou-se como principal alvo de avaliação a coerência na informação partilhada pelos três fluxos associados ao servidor *Python*. Na verdade, é crucial averiguar se o que está a ocorrer no *shop floor* é fielmente replicado tanto na base de dados, como no modelo 3D desenvolvido com o *software FlexSim*.

Neste sentido, procurou-se em momentos diferentes - e consequentemente em pontos de situação do *shop floor* diferentes - fazer uma análise de coerência entre o que está a ser enviado pelo emulador da fábrica, a informação dada pela BD e a distribuição das peças no modelo do sistema.

4.4.1 *Layout job shop*

O tipo de produção considerado para a implementação da monitorização em tempo real é um *job shop* que é constituído por sete estações de trabalho WS*x*, em que *x* corresponde ao número identificador único da estação entre 1 e 7. Cada WS tem capacidade de 1 peça e a disposição das mesmas é detalhada na Tabela 4.2. Apesar da elevada flexibilidade deste *layout* e da alta personalização de peças, os tempos de transporte não são considerados e as peças são movimentadas de forma automática entre os elementos que compõem a fábrica.

Tabela 4.2: Dados das estações de trabalho

Estação	PosX	PosY	Capacidade
WS1	-4.5	6.0	1
WS2	1.5	12.0	1
WS3	7.0	6.0	1
WS4	1.5	0	1
WS5	-4.5	-6.0	1
WS6	1.5	-12.0	1
WS7	7.0	-6.0	1

Adicionalmente, cada WS é antecedida por um *buffer* de entrada, B*x*, que ao contrário das estações de trabalho não apresentam uma capacidade fixa. Deste modo, as peças a ser processadas devem aguardar nos *buffers* antes de darem entrada na WS respetiva, caso esta esteja ocupada. A partir daqui, têm um comportamento de acordo com o método FIFO (*First-In, First-Out*) que garante que a primeira peça a entrar em B*x* será a primeira a ser transportada para WS*x*.

Por fim, existem ainda os elementos *Source* (ou *Queue1* no modelo) e *Sink*. O primeiro é responsável pela criação das peças, ao passo que o segundo fica encarregue da remoção das peças do modelo.

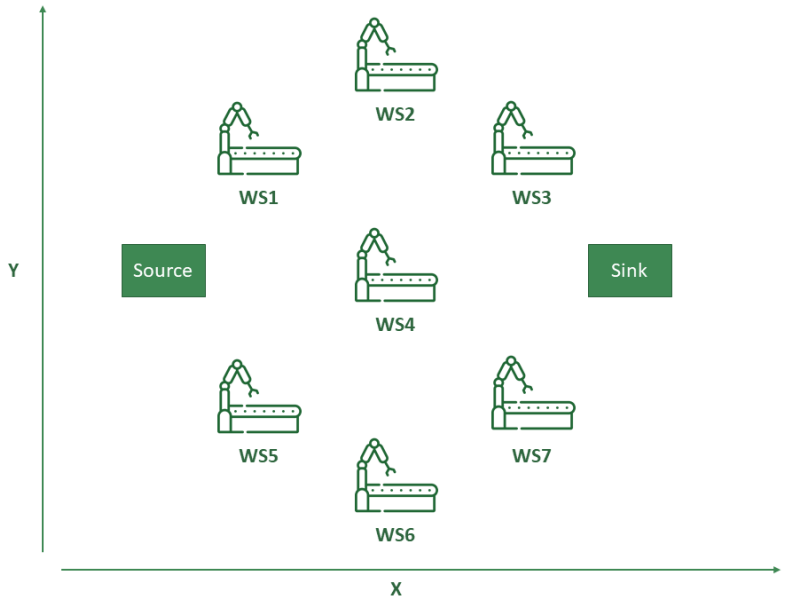

Figura 4.14: *Layout job shop* a criar no modelo 3D

4.4.2 Resultados

No âmbito de apurar a coerência transversal a todos os constituintes da *framework*, foram escolhidos instantes de forma aleatória e analisou-se nesse momento se a informação em cada uma das partes era a mesma.

Considere-se então o instante t=12s. Observando a Tabela 4.3, que contém um excerto da emulação da movimentação na fábrica, é evidenciado a cor azul que, nesse instante, a peça A1 deve estar em processamento na WS2 enquanto que a peça A2 deve estar em processamento na WS7. Resta então compreender se aquando do envio destes eventos pelo servidor, a base de dados e o modelo 3D apresentam os mesmos resultados.

Tabela 4.3: Excerto relevante do ficheiro de emulação de eventos utilizado

TOO	DESC	APA	AA
...	...	...	...
10	ON_ENTER	WS2	A1
11	ON_FINISH	WS1	A2
11	ON_EXIT	WS1	A2
11	ON_ENTER	B7	A2
11	ON_EXIT	B7	A2
11	ON_ENTER	WS7	A2
17	ON_FINISH	WS2	A1
...	...	...	...

De facto, as Figuras 4.15 e 4.16 que contêm a informação em ambas as coleções da BD, confirmam que no instante em estudo a informação na mesmas corresponde aos eventos que estão a ocorrer (gerados pelo ficheiro de emulação referido). A WS2 está a processar a peça A1, a WS7 está a processar a peça A2, e as restantes estão paradas - *"Idle"* - ao passo que todos os *buffers* estão vazios - *"Empty"*.

objects

	_id String	Status String	Agent Array	0 String
1	"WS1"	"Idle"	[] 0 elements	No field
2	"WS2"	"Processing"	[] 1 elements	"A1"
3	"WS3"	"Idle"	[] 0 elements	No field
4	"WS4"	"Idle"	[] 0 elements	No field
5	"WS5"	"Idle"	[] 0 elements	No field
6	"WS6"	"Idle"	[] 0 elements	No field
7	"WS7"	"Processing"	[] 1 elements	"A2"
8	"B1"	"Empty"	[] 0 elements	No field
9	"B2"	"Empty"	[] 0 elements	No field
10	"B3"	"Empty"	[] 0 elements	No field
11	"B4"	"Empty"	[] 0 elements	No field
12	"B5"	"Empty"	[] 0 elements	No field
13	"B6"	"Empty"	[] 0 elements	No field
14	"B7"	"Empty"	[] 0 elements	No field

Figura 4.15: Coleção *objects* no instante t=12s

agents

	_id String	Location String	Status String
1	"A1"	"WS2"	"Processing"
2	"A2"	"WS7"	"Processing"

Figura 4.16: Coleção *agents* no instante t=12s

Outro aspeto que corrobora a sincronização entre a BD e o servidor *Python* - e consequentemente o ficheiro de emulação - é o facto de as coleções *objects* e *agents* terminarem com os objetos vazios ou parados e sem quaisquer peças existentes, respetivamente. Na verdade, o ficheiro CSV de emulação da fábrica termina com a remoção de todas as peças do *shop floor*, levando a que a BD fique completamente limpa.

Por fim, pretende-se averiguar a situação no modelo 3D do *FlexSim*. Efetivamente, a Figura 4.17 demonstra, para o instante assinalado em *Run Time* igual a 12.33s, que existem de facto duas peças a ser processadas nas estações WS2 e WS7, sugerindo que este modelo segue também fielmente as instruções enviadas pelo servidor.

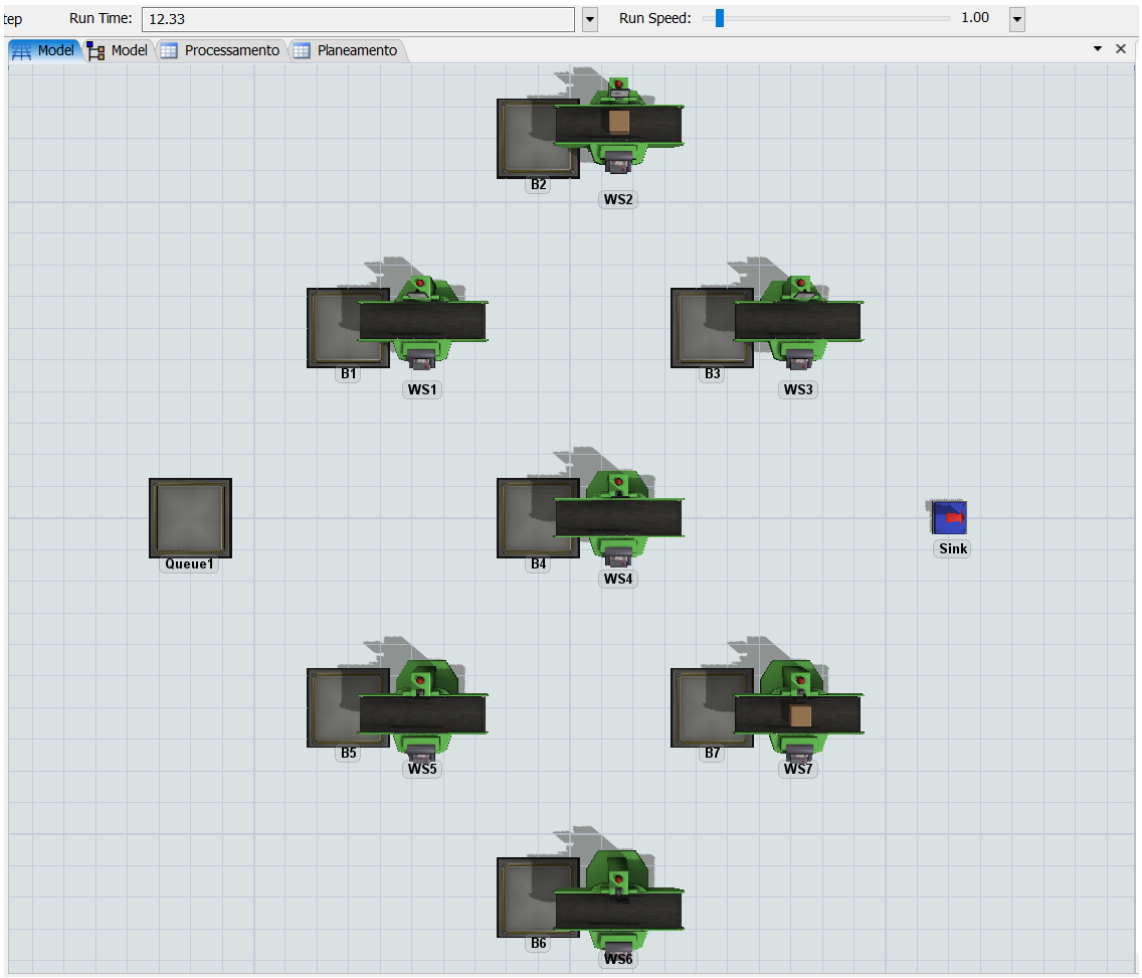

Figura 4.17: Modelo 3D no instante t=12s

Posto isto, os resultados mostram ter sido possível integrar de forma coerente os vários componentes. Na verdade, há interoperabilidade de dados nos três fluxos de informação, desde os eventos gerados pelo produtor à informação armazenada na BD e ao modelo visual no *FlexSim*.

Capítulo 5

Simulação

A capacidade de visualização e análise em tempo real do *shop floor* é sem dúvida uma característica valiosa do *digital twin* desenvolvido. Não obstante, no âmbito de ir além do presente e possibilitar o estudo do comportamento da fábrica no futuro, procurou-se incorporar uma vertente de simulação de eventos discretos que tem por base planos de produção agendados pelo gestor de produção.

De forma semelhante à componente de monitorização em tempo real abordada no Capítulo 4, recorre-se sobretudo ao *software FlexSim* para a realização desta tarefa. Contudo, a simulação pode ser feita *offline*, sem necessitar de uma comunicação com o servidor. Deste modo, existem dois aspetos distintos a considerar: a simulação de um plano de produção de forma isolada, que pode ser feita sem ligação ao servidor, e a simulação em paralelo com os eventos que ocorrem em tempo real, que requer uma ligação ao servidor. Ambos os modos de simulação necessitam, no entanto, de uma Tabela de Planeamento - que deve ser preenchida com as ordens de produção - e de uma Tabela de Processamento - que deve conter a rota pelas estações de trabalho de cada tipo de peça.

5.1 Processamento das peças

O processamento das peças é suportado por uma Tabela de Processamento, do qual é exemplo a Tabela 5.1, que deve conter a rota a percorrer por cada peça de um dado tipo. De facto, o percurso de cada tipo de peça deve ser pré-configurado pelo utilizador, de acordo com as necessidades da peça. A título exemplificativo e conforme demonstrado na Tabela referida, as peças do tipo 1 são inicialmente processadas na estação WS7 e terminam na estação WS4. Cada peça contém um apontador para a linha em que se encontra nesta Tabela, a *label* "linha" associada à *token* da peça, e à medida que a peça avança de WS essa é incrementada, como mencionado em 4.3.1. Por outras palavras, quando uma peça do tipo 1 deixa a WS7 a sua *label* "linha" é incrementada indicando que o próximo estágio da peça passa pela WS3.

Tabela 5.1: Exemplo de Tabela de Processamento

Linha	Tipo Peça	WS	PT (min)
1	1	WS7	3
2	1	WS3	7
3	1	WS2	7
4	1	WS4	5
5	2	WS2	2
6	2	WS1	9
7	2	WS7	7
8	2	WS6	8

Evidentemente, para tipos de processos diferentes a sequência de estações de trabalho deve seguir uma outra ordem. A Tabela 5.2 demonstra como o processamento de uma peça poderia ser implementado para processos do tipo *flow shop*, caracterizados por um *layout* em linha onde as peças passam pelas estações de trabalho por ordem numérica ascendente, isto é, de WS0 a WS4.

Tabela 5.2: Exemplo de Tabela de Processamento para processos do tipo *flow shop*

Linha	Tipo Peça	WS	PT (min)
1	1	WS0	3
2	1	WS1	1
3	1	WS2	3
4	1	WS3	7
5	1	WS4	8

5.2 Plano de produção

Por outro lado, caso o utilizador pretenda fazer a simulação de um plano de produção por si definido, este deve atualizar a Tabela de Planeamento que é o ponto de partida para a execução da simulação. As peças dão entrada no sistema no instante correspondente ao *Release Time* e são processadas de acordo com o seu tipo, através de uma consulta à Tabela de Processamento 5.1, vista anteriormente. Na Tabela 5.3 é apresentado um exemplo de planeamento possível.

Tabela 5.3: Exemplo de Tabela de Planeamento da produção

Linha	Release Time	ID Ordem de Produção	Tipo Peça
1	0	1	1
2	0	2	2
3	0	3	1
4	1	4	1
5	4	5	2
6	5	6	2
7	20	7	1

5.3 Processamento no modo de Simulação - *FlexSim*

O modo de simulação no *FlexSim* é composto por três partes essenciais: a procura por peças em processamento no momento do início da simulação, a criação das peças de acordo com as ordens de produção e o processamento das peças em concreto. A Figura 5.1 ilustra de forma sintetizada os passos principais desta componente.

A primeira responsabiliza-se por localizar as peças nos diversos *buffers* e estações de trabalho que existam no momento em que o utilizador dá início ao processo de simulação. Isto porque o sistema poder-se-ia encontrar no modo de monitorização, sendo necessário identificar os tipos das peças na fábrica e o estado de processamento em que as mesmas se encontram.

Depois, devem ser criadas as peças a processar, numa fila inicial, em conformidade com o plano que foi submetido pelo utilizador. A Tabela de Planeamento, de que é exemplo a Tabela 5.3, é lida desde a sua primeira à sua última linha e vão sendo criadas as peças do respetivo tipo no instante listado.

Por fim, resta fazer a lógica de movimentação. As peças têm de ser recebidas, processadas e enviadas nos elementos da fábrica conforme a informação dada pela Tabela de Processamento. Ao contrário da tarefa de monitorização que tem a capacidade de ser corrida indefinidamente, a tarefa de simulação termina após todas as peças do plano serem processadas.

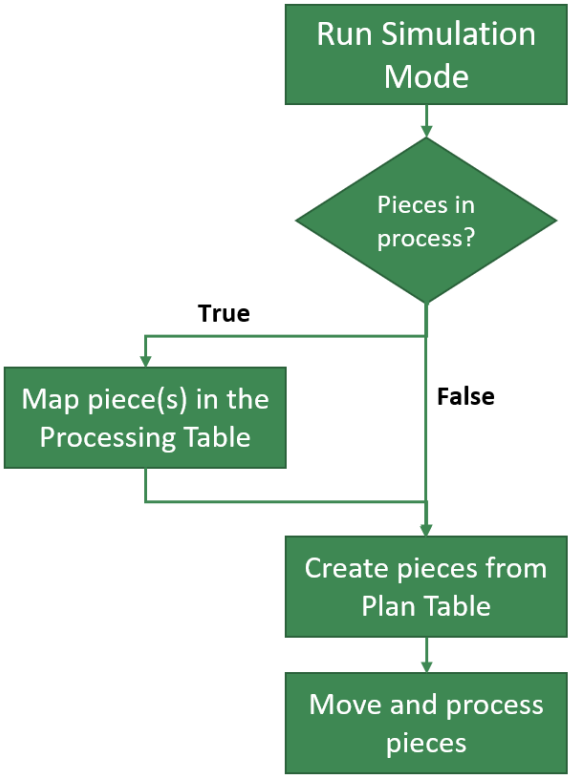

Figura 5.1: Diagrama representativo dos passos essenciais ao modo de simulação

5.3.1 Localização de peças em processamento

O processo de simulação é iniciado no momento em que o utilizador usa o botão *"Simulation Mode"*, situado no *dashboard* do *FlexSim*. Nesse instante, uma variável global *simulation_request* é colocada a 1, forçando uma outra variável global *simulation_start* a 1 também e terminando a atuação no fluxograma principal da Figura 4.10, dado que a *token* é descartada no bloco *Sink*. Na verdade, a aplicação de duas variáveis globais permite que o modo de simulação apenas tenha início na próxima iteração em que não existe uma mensagem recebida - *NO_MESSAGE* - evitando conflitos com a tarefa de monitorização. Como a simulação tem início na mudança de estado de uma variável, convém esta última não estar diretamente ligada ao comando do utilizador, visto que este comando pode ser dado a qualquer momento, inclusive a meio do processamento de mensagens na monitorização como referido.

Inicializando a simulação, o sistema deve começar por procurar e localizar peças que estejam em processamento no *shop floor*. Deste modo, como demonstrado no fluxograma de processos na Figura 5.2 devem ser percorridos todos os *buffers* da fábrica (no ramo à esquerda) e todas as *workstations* (no ramo à direita), a fim de averiguar a existência de peças antes de se iniciar a simulação. A lógica de pesquisa é a mesma tanto para um caso como para outro e culmina somente quando todos os *buffers* e todas as WS foram percorridas, mecanismo assegurado pelo bloco *Batch* que exige que duas *tokens* cheguem até si para prosseguir - a *token* que pesquisa nos *buffers* e a *token* que pesquisa nas WS.

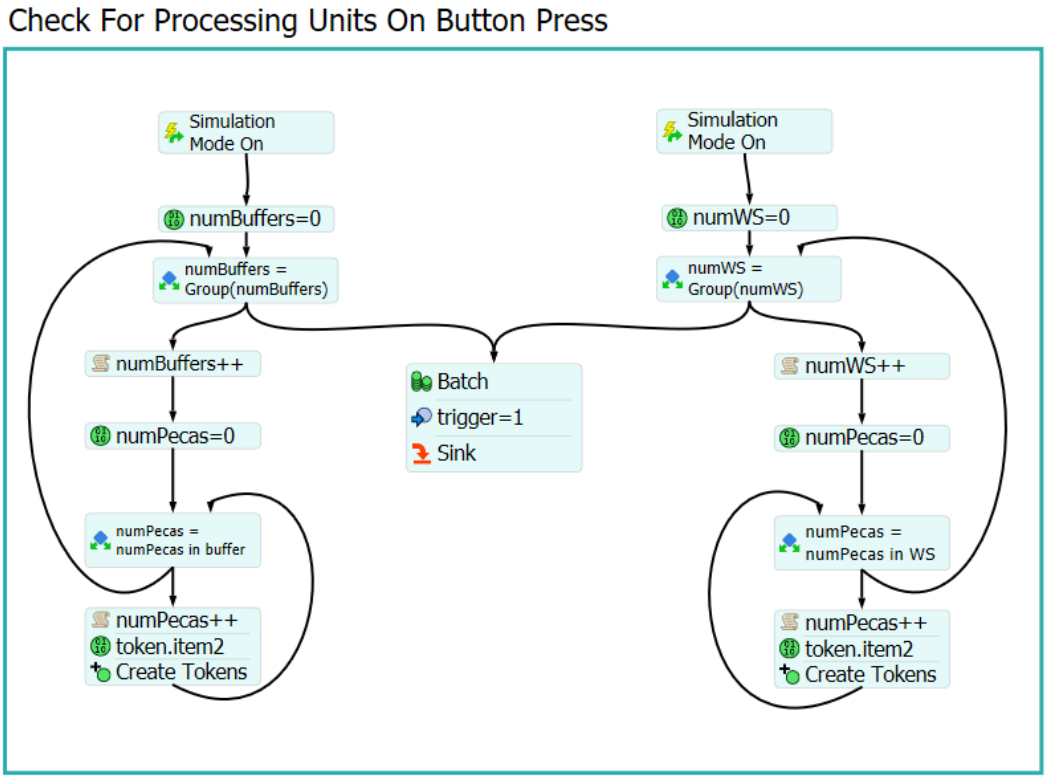

Figura 5.2: Localização de Peças em Processamento

Este processo começa por associar à *token* a *label "numBuffers"* ou *"numWS"*, com valor 0, que vai armazenar o número de elementos que já foram percorridos para a pesquisa de peças. Esta associação é feita com um bloco *Assign Labels* que atribui uma *label* a uma *token* com o valor inicial desejado. Enquanto ainda houver elementos a verificar, a *label* é incrementada em 1 a cada iteração, o que permite passar ao próximo *buffer* ou WS dependendo do caso. A cada iteração, a *label* começa então por ser incrementada e imediatamente a seguir é criada uma outra *label, "numPecas",* de forma idêntica às restantes *labels*. Se o número de peças no objeto em análise for superior ao número indicado pela *label "numPecas"*, esta é incrementada e são criadas *tokens* - recorrendo ao bloco *Create Tokens* - no fluxograma de processamento das peças, abordado em 5.3.3, até o número de peças no objeto e na *label* corresponderem. Caso contrário, termina a iteração e a *token* volta a verificar se ainda existem elementos, chamados também de objetos, por averiguar.

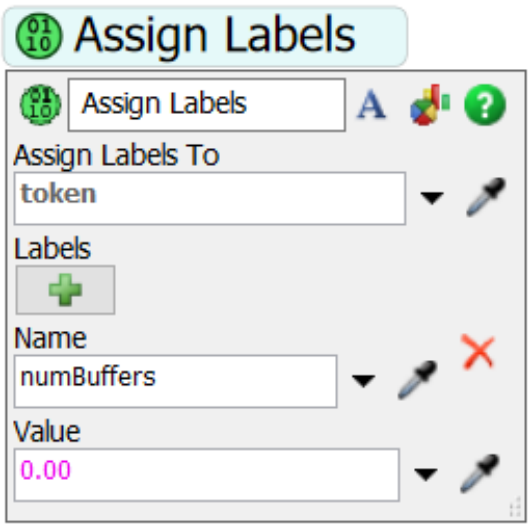

Figura 5.3: Bloco responsável por associar *labels* a uma *token*

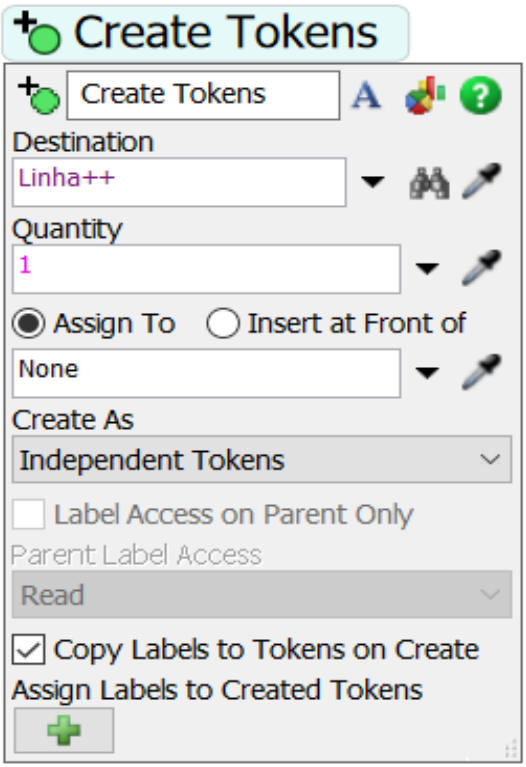

Figura 5.4: Bloco responsável por criar *tokens*

Terminada esta pesquisa tanto nos *buffers* como nas *workstations*, as *tokens* de ambos os ramos da Figura 5.2 entram no *Batch*, que aprova finalmente a continuação no fluxograma. A variável global *trigger* é colocada a 1, autorizando o início das duas outras etapas associadas ao modo de simulação: a criação das peças provenientes do plano de produção e o processamento e movimentação das peças no *shop floor*, que serão descritas a seguir. A *token* deste fluxograma é então apagada no bloco *Sink*, estando finalizada a fase de localização de peças em processamento no momento em que se deu início à simulação. Neste momento, as peças que estão em processamento estão mapeadas na Tabela de Processamento, conhecendo o resto do seu percurso.

5.3.2 Criação das peças a partir do plano de produção

A tarefa de criação das peças do plano de produção assegura, como o seu título indica, que são criadas no *FlexSim* as peças do plano introduzido pelo utilizador, no instante listado. Como já foi mencionado, este fluxograma apenas tem início após o desfecho da tarefa de localização de peças em processamento, abordada em 5.3.1, quando a variável global *trigger*=1 ou quando termina essa tarefa na *Sink*, desbloqueio realizado no bloco *Wait for Event*. Seguidamente, é atribuída à *token* a *label* "linha"com valor, que será um apontador para a linha atual da Tabela de Planeamento que está a ser processada. Deste modo, um bloco *Decide* verifica se o número da linha atual corresponde ao tamanho da tabela, e se for o caso termina o processamento deste fluxograma numa *Sink*. Caso contrário, se existirem ainda linhas não processadas, a *label* é incrementada, é implementado um *Delay* de duração igual à diferença dos *Release Times* e é criada uma *token* com o ID da ordem de produção e o tipo da peça no conjunto de blocos de baixo do fluxograma.

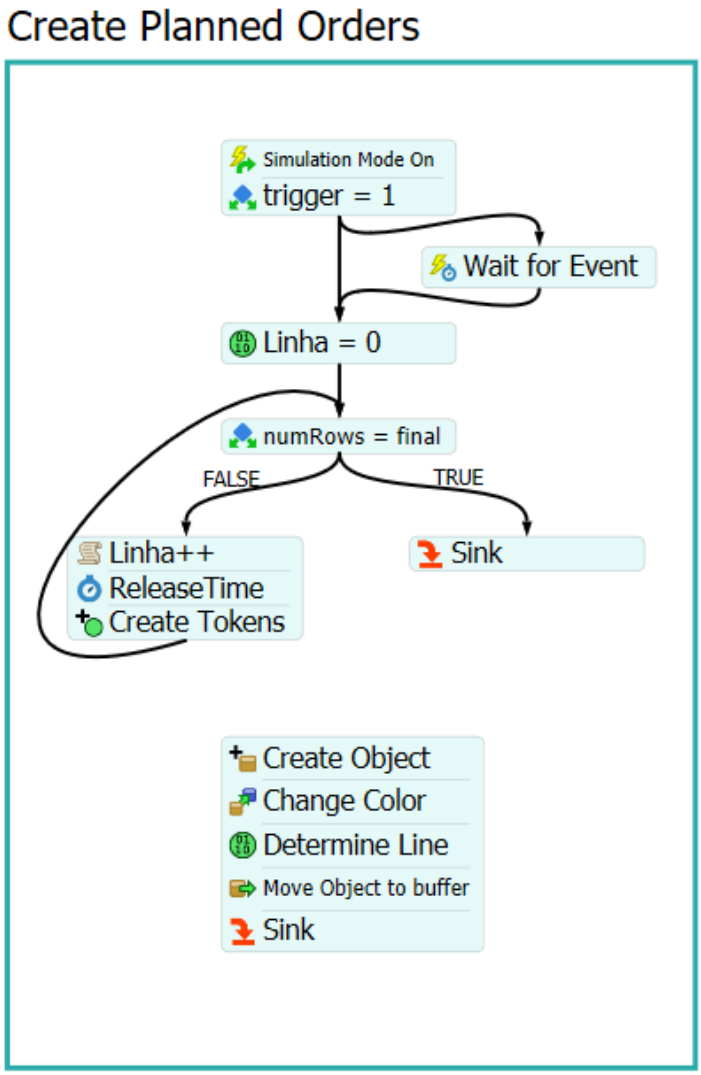

Figura 5.5: Criação das Peças do Plano de Produção

Tomando a linha 4 da Tabela 5.3 como exemplo, no bloco *Decide* o resultado será *FALSE* dado que ainda não se atingiu o fim da tabela - que tem tamanho 7 e não 4 - e a *label* é alterada para 5. O programa irá então fazer as diferenças entre os *Release Time* listados e aguardar 4-1=3 unidades de tempo, gerando depois uma *token* no conjunto de baixo da Figura 5.5, responsável pela criação de cada peça como será detalhado a seguir.

Como descrito previamente, a *token* no bloco *Create Object* contém informação sobre o seu tipo e sobre o identificador da ordem de produção da qual faz parte. É também atribuída uma cor à peça dependendo do respetivo tipo, de forma a facilitar a inspeção visual durante a fase de testes. Para além do tipo e da ordem de produção, é atribuída mais uma *label* que deve apontar para a linha na tabela de processamento do respetivo tipo. Dando seguimento ao exemplo do parágrafo anterior, à peça da linha 5 da Tabela 5.3 será concedida uma *label* com a linha 5 da Tabela 5.1, uma vez que é o primeiro passo para peças do tipo 2. Por fim, esta é movida para o *buffer* de entrada associado à primeira estação de trabalho em que vai ser processada a peça.

Esta tarefa termina quando o plano do utilizador, introduzido na tabela de planeamento, está totalmente processado e todas as peças do mesmo estão criadas no modelo 3D, à semelhança da Figura 5.6 onde todas as peças do plano acabam de ser criadas no *buffer* inicial. Posteriormente será feito o processamento das peças já existentes e das peças do plano geradas, etapa analisada na próxima secção.

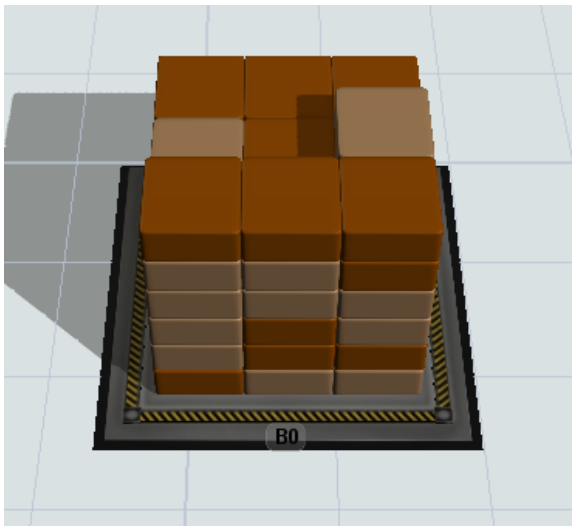

Figura 5.6: Grupo de peças do plano criadas no início da simulação

5.3.3 Processamento das peças

A tarefa de processamento, cujo fluxograma é ilustrado na Figura 5.7, refere-se à movimentação das peças criadas a partir do plano de produção, no modelo 3D. Tal como foi feito no Capítulo 4 relativo à monitorização, é necessário um fluxograma com a lógica de deslocação pelos objetos do *shop floor*. Não obstante, desta vez não é um programa externo a informar o *FlexSim*

da movimentação a fazer mas sim o próprio *FlexSim* a fazer esse processamento com auxílio da tabela de processamento onde é mapeado o estágio de cada peça.

De forma a restringir o número de peças numa WS, utilizou-se um outro recurso partilhado do *FlexSim*, denominado *Zone*. Cada zona é limitada a uma peça assegurando que, por conseguinte, enquanto uma WS estiver ocupada a entrada de mais entidades está bloqueada. Deve existir então uma zona para cada WS no *shop floor*.

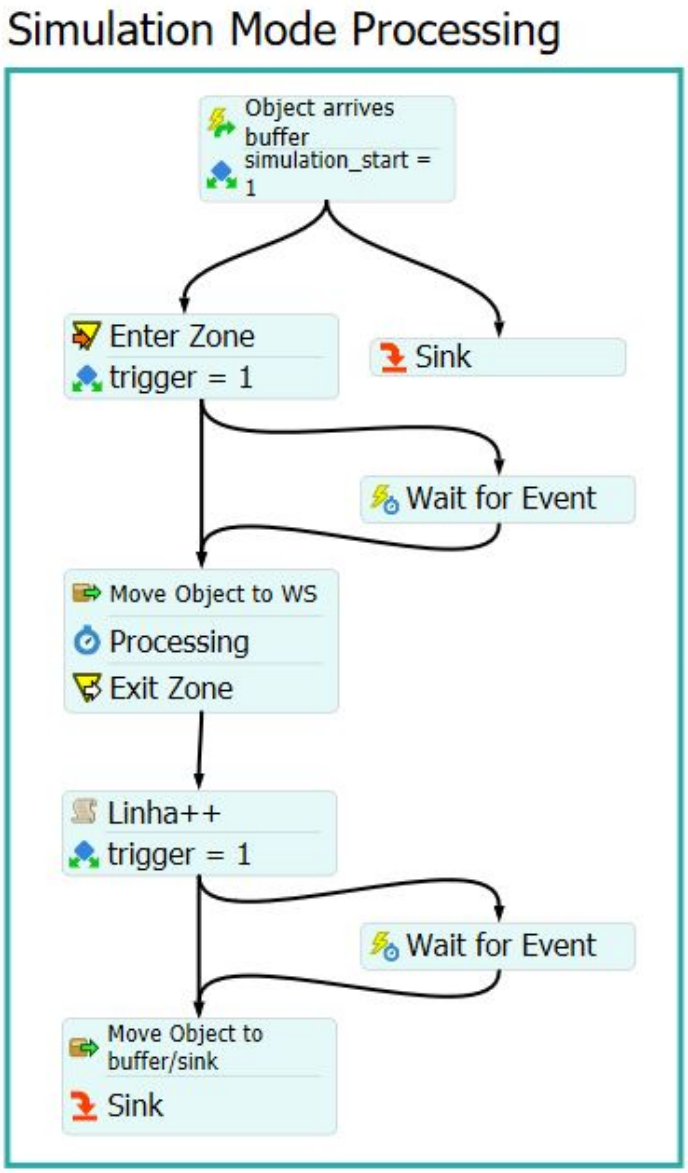

Figura 5.7: Processamento em Modo de Simulação

Com efeito, esta tarefa arranca sempre que uma peça entra num *buffer* mas apenas tem continuidade caso a variável *simulation_start* esteja com valor 1, ou seja, caso o utilizador tenha ativado

o modo de simulação. Segue-se a entrada na zona respetiva, que apenas ocorre se a mesma se encontrar vazia. É aqui que as *tokens* relativas às peças esperam numa fila FIFO até que a zona em que pretendem entrar fique livre. Se a variável *trigger* não estiver a 1, o bloco *Wait for Event* força uma pausa até que a tarefa de localização das peças termine. Só depois é criada e processada a entidade na WS correspondente, sendo retirada da zona no fim do processamento. Por fim, a *label* "linha", que aponta para a fase de processamento da peça na tabela de processamento, é incrementada e a peça é movida para o *buffer* da WS listada na nova linha ou apagada numa *Sink* caso a nova linha corresponda a um tipo de peça diferente, indicando que terminou o tratamento da peça.

5.4 Validação e resultados

A validação da componente de simulação recai sobre a concretização das três etapas principais, abordadas em 5.3.1, 5.3.2 e 5.3.3. Neste sentido, deve ser assegurado que no momento em que se dá início à simulação as peças em processamento não são apagadas e que todas as peças do plano de produção são introduzidas e processadas no modelo 3D. A validação do modo de simulação foi feita com base no *layout job shop* descrito em 4.4.1.

Recuperando o ficheiro de emulação da fábrica da tabela 4.3, será estudado uma vez mais o instante t=12s. Como visto na validação da componente de monitorização em 4.4.2, existem duas peças a ser processadas nesse momento: A1 em WS2 e A2 em WS7. Introduzindo o plano da Tabela 5.4 e dando início à simulação do mesmo nessa altura, deve ser dado seguimento ao processamento das peças A1 e A2 e criar em simultâneo as do plano. De facto, são introduzidas duas peças do tipo 1 e três do tipo 2 com *Release Time* 0, isto é, assim que se inicia a simulação. O processamento das peças do tipo 1 e 2 é idêntico ao detalhado na Tabela 5.1.

Tabela 5.4: Tabela de Planeamento introduzido para a validação

Linha	Release Time	ID Ordem de Produção	Tipo Peça
1	0	1	1
2	0	2	2
3	0	3	1
4	0	4	2
5	0	5	2

De forma a ser possível distinguir as peças provenientes da monitorização das peças do plano submetido, utilizou-se o seguinte código de cores:

Tabela 5.5: Código de cores para as peças

Tipo	Cor	#
Peças monitorização	Castanho	#987552
Novas peças tipo 1	Azul	#0000ff
Novas peças tipo 2	Vermelho	#ff0000

Analisando o comportamento do modelo, vemos na Figura 5.8 que, como esperado, existem três peças do tipo 2 (vermelho) prontas a entrar em WS2 e duas peças do tipo 1 (azul) prontas a entrar em WS7. Também as peças que estavam em monitorização (castanho) prosseguiram para a simulação, estando ambas em processamento.

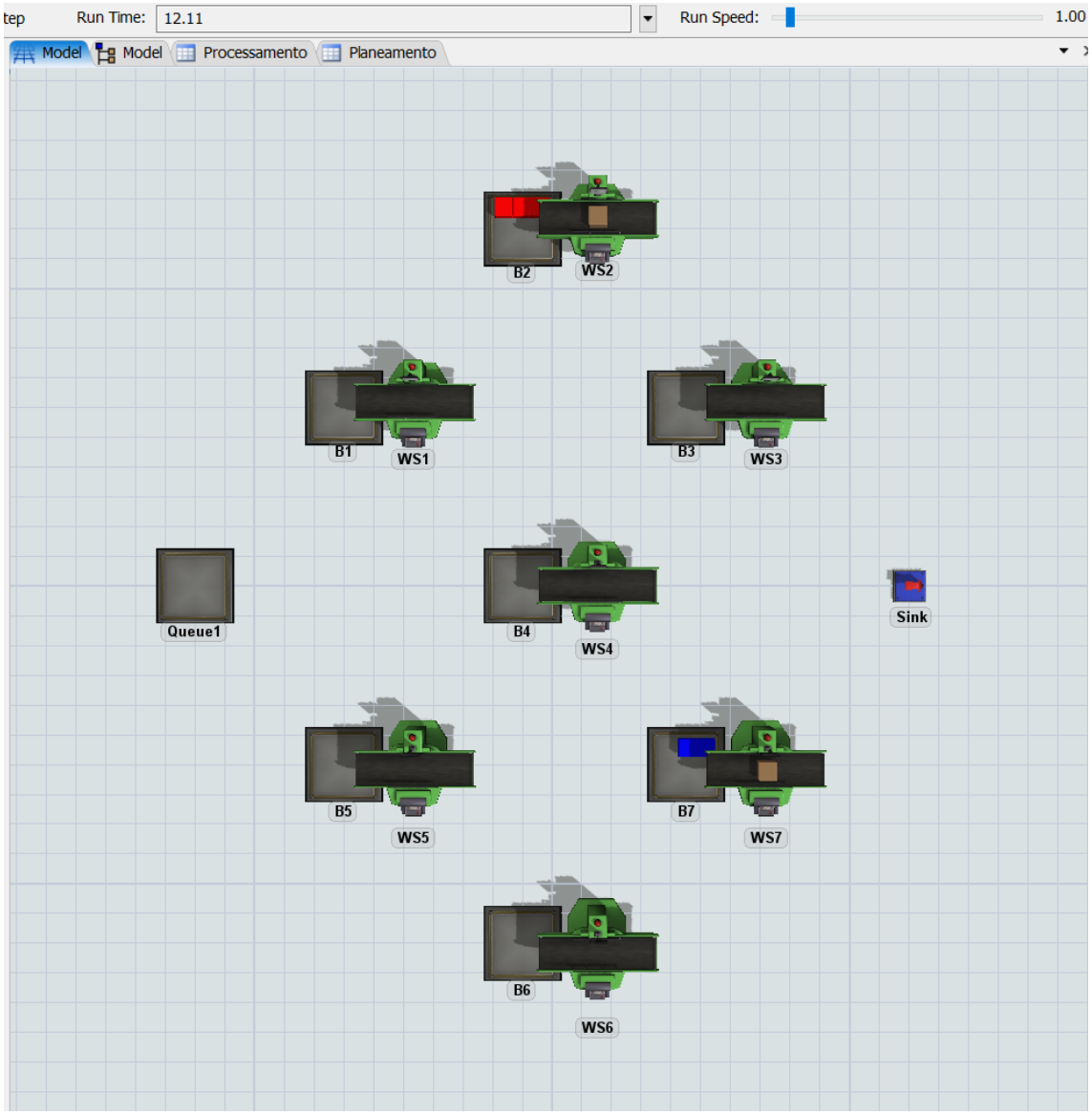

Figura 5.8: Modelo 3D no instante t=12s - Modo Simulação

Avançando na simulação, o processamento vai continuando e as peças vão-se movimentando no modelo de acordo com as suas sequências.

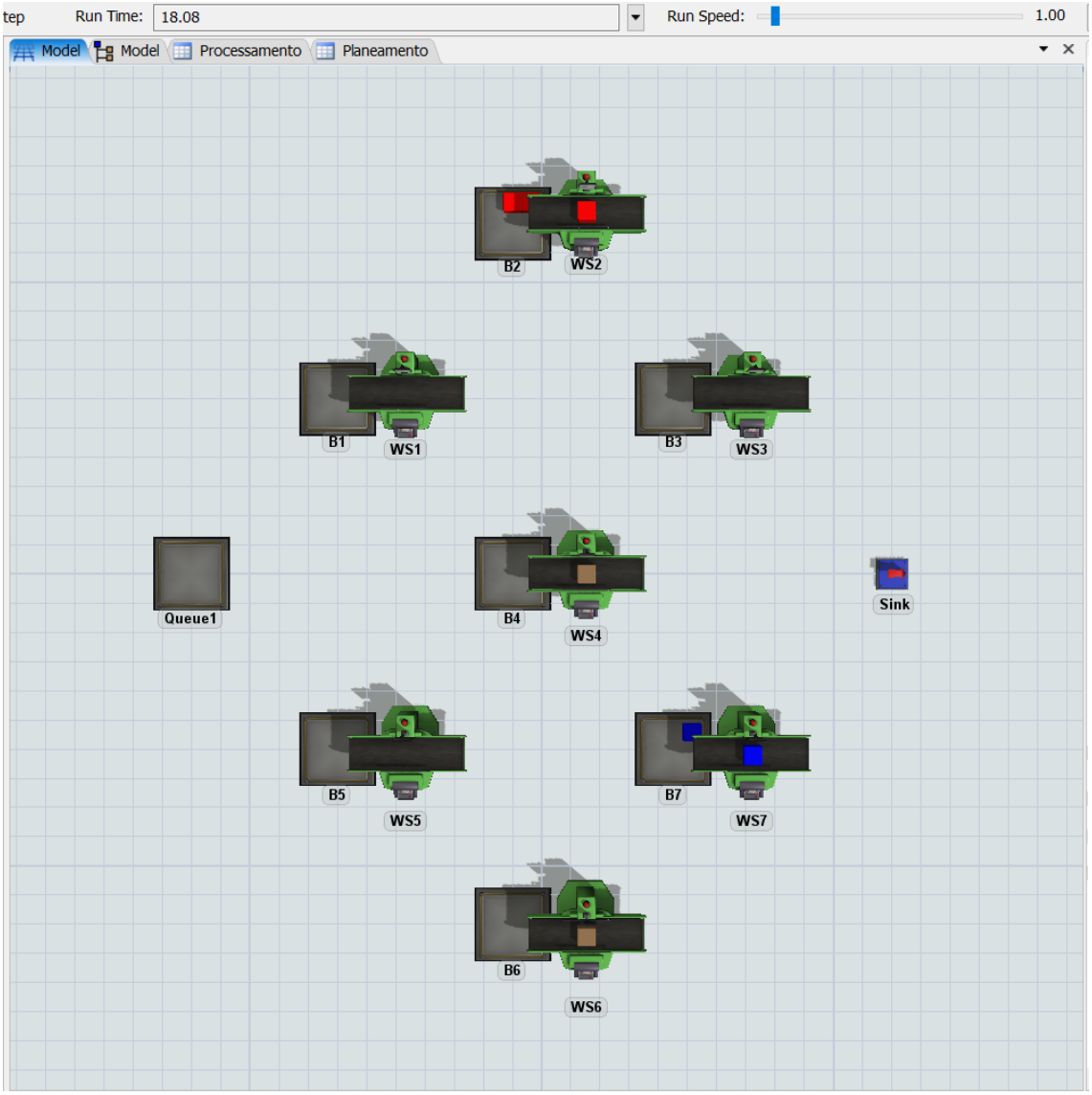

Figura 5.9: Modelo 3D no instante t=18s - Modo Simulação

Deste modo, é evidente que as peças resultantes da tarefa de monitorização continuam o seu processamento, enquanto que as peças do plano de produção são introduzidas e processadas em simultâneo. A ponte feita entre as componentes de monitorização e simulação consegue garantir que não há perda de informação graças à *snapshot* do último estado da fábrica, guardada na BD.

Capítulo 6

Otimização

O grande objetivo desta componente passa por demonstrar a interoperabilidade da plataforma desenvolvida. Na verdade, a capacidade de comunicar com serviços inteligentes externos abre inúmeras portas à otimização e ao constante aprimoramento dos processos. Neste capítulo será discutido o serviço a que se recorreu em particular, um programa de otimização de planos de produção. A partir do envio das tabelas de planeamento e de processamento mencionadas nos capítulos anteriores é possível receber uma sequência otimizada que minimiza o *makespan*, alterando apenas a sequência no plano.

6.1 Comunicação com serviços externos

A comunicação feita entre o *FlexSim* e o serviço externo é feita recorrendo a uma aplicação mediadora, uma vez mais através de *sockets* TCP/IP. Esta aplicação, chamada *External Services App* na Figura 6.1 que ilustra o fluxo de informação, recebe o plano que foi submetido pelo utilizador no *FlexSim* e reencaminha-o para um programa de otimização que se encontra noutro computador ou servidor. No sentido oposto, quando o plano está otimizado é enviado de volta para a aplicação mediadora, que remete para o *FlexSim* esta sequência otimizada. Esta sequência é essencialmente um vetor com a ordem em que devem ser processadas as peças, enviado no formato JSON.

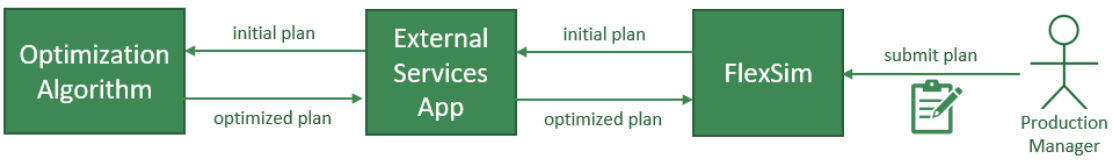

Figura 6.1: Esquema do fluxo de informação na tarefa de otimização

6.2 Otimização de planos de produção - *FlexSim*

No *FlexSim*, a *socket* é criada quando é corrido o modelo com auxílio do módulo *Model Trigger* denominado *OnRunStart*, previamente visto em 4.1.1. Efetivamente são criadas duas *sockets* com portas diferentes, uma dedicada à monitorização e outra para comunicação com a aplicação mediadora de serviços externos. O pequeno fluxograma da Figura 6.2 está encarregue de fazer o pedido de otimização, enviar o plano submetido, receber a solução e alterar a sequência na tabela de planeamento.

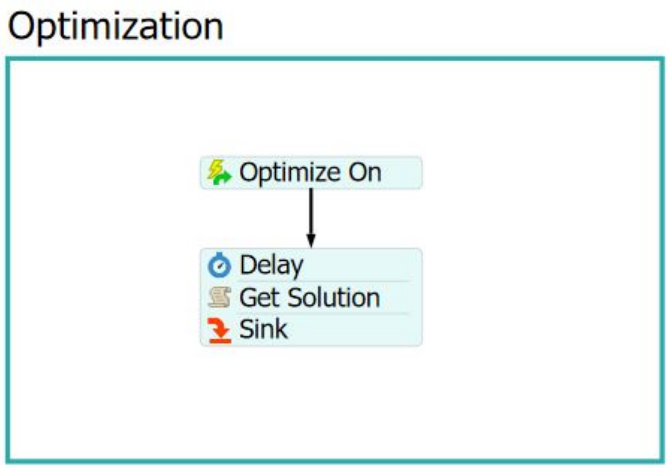

Figura 6.2: Fluxograma da otimização de planos

Inicialmente, o pedido de otimização é feito no botão da Figura 6.3, presente no *dashboard* do *FlexSim*. Um bloco *Delay* assegura que o *FlexSim* consegue processar este pedido. Depois, o bloco de código *Get Solution* trata de enviar à aplicação mediadora o pedido de otimização com o plano atual. Feito o pedido, o fluxograma fica bloqueado até receber a solução. Aquando da receção da mesma, é feito o *parsing* da mensagem JSON e a coluna com o tipo de peça da tabela de planeamento é atualizada com o vetor que contém a solução otimizada, como veremos na secção de resultados.

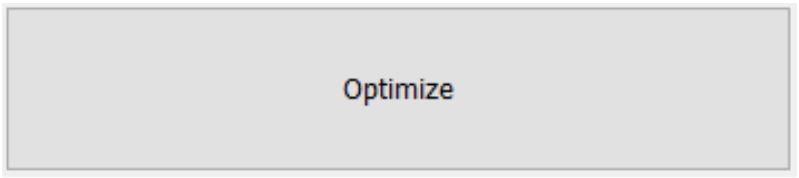

Figura 6.3: Botão de otimização

6.3 Validação e resultados

A validação da tarefa de otimização passa por analisar não só a mudança nas sequências de produção, mas também a variação do *makespan*. Com efeito, a otimização deve direcionar o plano de produção no sentido de minimizar o *makespan*, sendo esse o principal alvo desta funcionalidade. Adicionalmente, ao contrário das componentes de monitorização e simulação, a otimização é testada num *layout flow shop* a fim de certificar a flexibilidade da plataforma desenvolvida, capaz de se adaptar a sistemas de produção com características diferentes.

6.3.1 *Layout flow shop*

De forma semelhante ao *layout* do tipo *job shop*, o *flow shop* apresenta também uma *Source* e uma *Sink* para a entrada e saída de peças, *buffers* de entrada e as respetivas estações de trabalho, como demonstrado na Figura 6.4. Contudo, este sistema é constituído apenas por 5 estações, numeradas de 0 a 4, e apresenta uma disposição em linha. Sendo assim, os tipos de peça distinguem-se pelos tempos de processamento nas diferentes estações de trabalho.

Figura 6.4: *Layout flow shop* em estudo

A Tabela 6.1 apresenta as características das estações de trabalho no modelo 3D.

Tabela 6.1: Dados das estações de trabalho

Estação	PosX	PosY	Capacidade
WS0	-13.5	-1.25	1
WS1	-5.5	-1.25	1
WS2	2.5	-1.25	1
WS3	10.5	-1.25	1
WS4	18.0	-1.25	1

6.3.2 Resultados

Foi submetido um plano na tabela de planeamento de 50 peças de 50 tipos diferentes. A Figura 6.5 apresenta o modelo visual no início da simulação em que são criadas as peças e a primeira é processada.

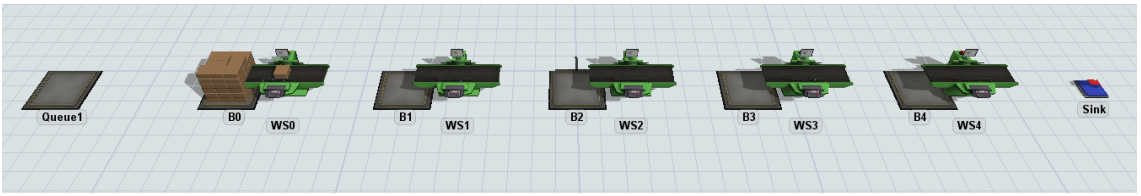

Figura 6.5: Modelo 3D no início da simulação do plano

A sequência de produção, por seu turno, foi gerada de forma aleatória e foi simulado o plano. Seguidamente, fez-se um pedido de otimização e, recebida a solução, simulou-se o novo plano da mesma forma. As Figuras 6.6 e 6.7 apresentam um excerto (primeiras 15 peças) do plano inicial e do plano otimizado, respetivamente.

Model | Processamento | Planeamento

	ReleaseTime	OrderID	Type
1	0	1	13
2	0	2	31
3	0	3	10
4	0	4	9
5	0	5	30
6	0	6	6
7	0	1	33
8	0	2	26
9	0	3	49
10	0	4	15
11	0	5	43
12	0	6	32
13	0	1	23
14	0	2	1
15	0	3	24

Figura 6.6: Excerto do Plano Inicial

Model | Processamento | Planeamento

	ReleaseTime	OrderID	Type
1	0	1	43
2	0	2	36
3	0	3	30
4	0	4	27
5	0	5	11
6	0	6	40
7	0	1	10
8	0	2	49
9	0	3	8
10	0	4	35
11	0	5	5
12	0	6	24
13	0	1	32
14	0	2	44
15	0	3	34

Figura 6.7: Excerto do Plano Otimizado

Evidentemente, estas alterações refletiram-se no *makespan*. Os resultados para ambas as simulações são expostos na Tabela 6.2 e revelam melhorias significativas.

Tabela 6.2: Resultados para o *makespan*

Makespan Plano Inicial (s)	*Makespan* Plano Otimizado (s)	Melhoria (%)
3418.40	3197.82	6.45

Capítulo 7

Conclusão e Trabalho Futuro

O presente capítulo dedica-se à discussão das conclusões referentes ao trabalho desenvolvido e à verificação do cumprimento dos objetivos estabelecidos. Adicionalmente, são abordadas potenciais implementações que se possam revelar de aplicação interessante num trabalho futuro.

7.1 Conclusões

O principal objetivo do projeto recaía sobre o desenvolvimento de uma *framework* que explorasse as capacidades de *Digital Twins* de processos e que permitisse a sua aplicação a diferentes tipos de produção numa fábrica. Esta *framework* deveria ser capaz de implementar um DT dos processos de um *shop floor*, que permite a monitorização da fábrica em tempo real, a simulação de planos de produção introduzidos pelo utilizador e a comunicação com serviços inteligentes externos, sendo que neste caso se recorreu a um programa de otimização de planos de produção.

Inicialmente, reconstruiu-se o *layout* fabril no ambiente de simulação, o *software FlexSim*. Foi considerado um processo *job shop* em que os tipos de peças são processados em sequências diferentes pelas estações de trabalho. É neste modelo tridimensional que se cria a vertente visual do DT através da monitorização em tempo real. O comportamento real na fábrica é emulado por mensagens guardadas num ficheiro CSV, que contêm o instante em que as peças entram na fábrica, o identificador único da peça, o tipo de evento (entrada, saída ou fim do processamento) e a estação de trabalho associada. Estas mensagens alimentam o servidor que as trabalha, enviando a informação necessária para a base de dados e para o *FlexSim*. Na BD é armazenada uma *snapshot* do último estado da fábrica, enquanto que no *FlexSim* é atualizado o modelo visual que permite seguir o comportamento do *shop floor* em tempo real - é assim concretizado o processo de *twinning*, isto é, a criação do gémeo digital da fábrica.

Posteriormente, implementou-se uma tarefa de simulação de planos de produção no *FlexSim*. Esta componente dedica-se à simulação de eventos futuros baseados num plano de produção introduzido pelo utilizador e pode ser feita a partir do estado da fábrica na monitorização. A simulação permite ao utilizador testar diferentes configurações do seu *shop floor* ou diferentes sequências de produção a fim de encontrar melhores soluções ou de compreender melhor os seus processos,

tendo em conta que o *FlexSim* fornece informação sobre diversos parâmetros como são exemplos o *makespan*, tempo que decorre do início ao fim do processo, ou o *staytime*, tempo que uma peça fica num dado *buffer* de entrada em espera. Nesta componente recorreu-se ao *layout* criado na monitorização para análise de resultados.

Por fim, foi incorporada uma tarefa de comunicação com serviços inteligentes, que neste projeto se tratou de um programa de otimização de planos de produção. Esta componente confere à *framework* interoperabilidade, uma vez que possibilita a cooperação com serviços para além do projeto desenvolvido. No que concerne ao programa de otimização em concreto, este tem a capacidade de encontrar sequências de produção do plano introduzido pelo utilizador que levem a menores *makespans*. Esta redução possibilita o processamento de um maior número de peças, que resulta numa maior produtividade da fábrica, como mostram os resultados obtidos para esta componente. Deve ser referido que, ao contrário da monitorização e da simulação, esta componente é aplicada a um processo *flow shop* caracterizado pelo *layout* em linha, por forma a demonstrar a flexibilidade e adaptabilidade da *framework* a diferentes tipos de processos.

Em conclusão, os objetivos da dissertação foram cumpridos na sua totalidade e o resultado deste projeto poderá ser uma base para trabalhos futuros no que diz respeito a *Digital Twins* de processos. A *framework* desenvolvida está aberta a ajustes e a novas funcionalidades, sendo um ponto de partida para a criação de uma plataforma funcional ao nível industrial.

7.2 Trabalho Futuro

Numa filosofia de melhoria contínua, torna-se pertinente estudar soluções no sentido de aprimorar a *framework* desenvolvida. Neste sentido, são apontados alguns tópicos eventualmente relevantes à elevação do projeto.

Um dos aspetos taxativos da plataforma desenvolvida é a inexistência de um sistema físico a ser replicado pelo *Digital Twin*. Efetivamente, o sistema real é emulado por sequências de mensagens, surgindo o interesse em estender a arquitetura do *shop floor* a um protótipo constituído por sensores e atuadores. Este seria o primeiro passo na integração da *framework*, cuja implementação se pretende alongar a um caso real mais tarde.

Um outro ponto que pode ser alvo de melhoria é o tempo de processamento nas estações de trabalho. Expandir a natureza unidimensional deste parâmetro para distribuições de probabilidade permitiria aproximar o modelo atual à realidade, dado que em geral o tempo de processamento de peças do mesmo tipo é raramente igual. Na verdade, ter em consideração esta incerteza garante que se têm em conta cenários menos esperados, baseados em dados probabilísticos e não apenas num valor constante.

Por fim, incluir recursos de transporte no motor de simulação pode também ser uma mais valia para a *framework*. Desde empilhadores a AGVs, incorporar métodos de transporte acrescenta realismo ao modelo desenvolvido, que deixa de ficar limitado a *buffers* e *workstations*. Esta melhoria contribui essencialmente para uma maior precisão no cálculo do *makespan* durante a simulação

de planos de produção do utilizador, dado que inclui tempos de transporte que estão presentes em qualquer sistema físico real.

Referências

[1] Apollo Spacecraft Program Office NASA Test Division. Apollo 13 technical air-to-ground voice transcriptor p. 167, nasa, Abril 1970. Disponível em https://www.hq.nasa.gov/alsj/a13/a13trans.html.

[2] Filomena Naves. Odisseia da apollo 13 foi há meio século, Abril 2020. Disponível em https://www.dn.pt/vida-e-futuro/odisseia-da-apollo-13-foi-ha-meio-seculo--12057700.html.

[3] Pieter van Schalkwyk. Digital twins: The ultimate guide. Disponível em https://xmpro.com/digital-twins-the-ultimate-guide/.

[4] Jay Lee, Edzel Lapira, Behrad Bagheri, e Hung-An Kao. Recent advances and trends in predictive manufacturing systems in big data environment. *Manufacturing Letters*, 1:38–41, 10 2013. doi:10.1016/j.mfglet.2013.09.005.

[5] Kagermann Henning. Recommendations for implementing the strategic initiative industrie 4.0. 2013.

[6] M. Brettel, Niklas Friederichsen, M. Keller, e N. Rosenberg. How virtualization, decentralization and network building change the manufacturing landscape: An industry 4.0 perspective. *International Journal of Science, Engineering and Technology*, 8:37–44, 08 2014.

[7] Daniel Engels, Sanjay Sarma, Laxmiprasad Putta, e David Brock. The networked physical world system. páginas 104–111, 01 2002.

[8] Elisa Negri, Luca Fumagalli, e Marco Macchi. A review of the roles of digital twin in cps-based production systems. *Procedia Manufacturing*, 11:939–948, 12 2017. doi:10.1016/j.promfg.2017.07.198.

[9] Keliang Zhou, Taigang Liu, e Lifeng Zhou. Industry 4.0: Towards future industrial opportunities and challenges. páginas 2147–2152, 08 2015. doi:10.1109/FSKD.2015.7382284.

[10] Shiyong Wang, Jiafu Wan, Di Li, e Chunhua Zhang. Implementing smart factory of industrie 4.0: An outlook. *International Journal of Distributed Sensor Networks*, 2016:1–10, 01 2016. doi:10.1155/2016/3159805.

[11] Scott Kennedy. Made in china 2025, Jun 2015. URL: https://www.csis.org/analysis/made-china-2025.

[12] Industry 4.0: fourth industrial revolution guide to industrie 4.0. URL: https://www.i-scoop.eu/industry-4-0/.

[13] Kyung-Joon Park, Rong Zheng, e Xue Liu. Cyber-physical systems: Milestones and research challenges. *Computer Communications*, 36:1–7, 12 2012. doi:10.1016/j.comcom.2012.09.006.

[14] Ragunathan Rajkumar, Insup Lee, Lui Sha, e John Stankovic. 44.1 cyber-physical systems: The next computing revolution. páginas 731–736, 01 2010. doi:10.1145/1837274.1837461.

[15] Laszlo Monostori, Botond Kádár, Thomas Bauernhansl, Shinsuke Kondoh, Soundar Kumara, Gunther Reinhart, Olaf Sauer, Günther Schuh, Wilfried Sihn, e K. Ueda. Cyber-physical systems in manufacturing. *CIRP Annals - Manufacturing Technology*, 65:621–641, 08 2016. doi:10.1016/j.cirp.2016.06.005.

[16] Kevin Ashton. That'internet of things' thing in thereal world things matter more than ideas. Em *RFID Journal*. 2009.

[17] Jorge E. Ibarra-Esquer, Félix F. González-Navarro, Brenda L. Flores-Rios, Larysa Burtseva, e María A. Astorga-Vargas. Tracking the evolution of the internet of things concept across different application domains. *Sensors*, 17(6), 2017. URL: https://www.mdpi.com/1424-8220/17/6/1379, doi:10.3390/s17061379.

[18] Mark Weiser. The computer for the 21 st century. *Scientific American*, 265(3):94–105, 1991. URL: http://www.jstor.org/stable/24938718.

[19] Jim Chase. The evolution of the internet of things, Sep 2013. URL: https://www.ti.com/lit/pdf/swrb028.

[20] The internet of things (iot) – essential iot business guide. URL: https://www.i-scoop.eu/internet-of-things-guide/.

[21] Hermann Kopetz. *Real-time systems: design principles for distributed embedded applications*. Springer Science & Business Media, 2011.

[22] Daniele Miorandi, Sabrina Sicari, Francesco De Pellegrini, e Imrich Chlamtac. Internet of things: Vision, applications and research challenges. *Ad Hoc Networks*, 10(7):1497 – 1516, 2012. URL: http://www.sciencedirect.com/science/article/pii/S1570870512000674, doi:https://doi.org/10.1016/j.adhoc.2012.02.016.

[23] Mahmoud Elkhodr, Seyed Shahrestani, e Hon Cheung. The internet of things: Vision & challenges. Em *IEEE 2013 Tencon-Spring*, páginas 218–222. IEEE, 2013.

[24] Azad M. Madni, Carla C. Madni, e Scott D. Lucero. Leveraging digital twin technology in model-based systems engineering. *Systems*, 7(1), 2019. URL: https://www.mdpi.com/2079-8954/7/1/7, doi:10.3390/systems7010007.

[25] Michael Grieves e John Vickers. *Digital Twin: Mitigating Unpredictable, Undesirable Emergent Behavior in Complex Systems*, páginas 85–113. 08 2017. doi:10.1007/978-3-319-38756-7_4.

[26] Mike Shafto, M Conroy, R Doyle, E Glaessgen, C Kemp, J LeMoigne, e L Wang. Modeling, simulation, information technology and processing roadmap. 05 2010.

[27] Michael Gichane, Jean Byiringiro, Andrew Chesang, Peterson Nyaga, Rogers Langat, Hasan Smajic, e Consolata Kiiru. Digital triplet approach for real-time monitoring and control of an elevator security system. *Designs*, 4:9, 04 2020. doi:10.3390/designs4020009.

[28] Werner Kritzinger, Matthias Karner, Georg Traar, Jan Henjes, e Wilfried Sihn. Digital twin in manufacturing: A categorical literature review and classification. *IFAC-PapersOnLine*, 51:1016–1022, 01 2018. doi:10.1016/j.ifacol.2018.08.474.

[29] Roland Rosen, Georg Wichert, George Lo, e Kurt Bettenhausen. About the importance of autonomy and digital twins for the future of manufacturing. *IFAC-PapersOnLine*, 48:567–572, 12 2015. doi:10.1016/j.ifacol.2015.06.141.

[30] Ján Vachálek, Lukás Bartalský, Oliver Rovný, Dana Šišmišová, Martin Morháč, e Milan Lokšík. The digital twin of an industrial production line within the industry 4.0 concept. Em *2017 21st International Conference on Process Control (PC)*, páginas 258–262, 2017. doi:10.1109/PC.2017.7976223.

[31] B. R. Barricelli, E. Casiraghi, e D. Fogli. A survey on digital twin: Definitions, characteristics, applications, and design implications. *IEEE Access*, 7:167653–167671, 2019. doi:10.1109/ACCESS.2019.2953499.

[32] Sebastian Haag e Reiner Anderl. Digital twin – proof of concept. *Manufacturing Letters*, 15, 02 2018. doi:10.1016/j.mfglet.2018.02.006.

[33] Guodong Shao, Sanjay Jain, Christoph Laroque, Loo Hay Lee, Peter Lendermann, e Oliver Rose. Digital twin for smart manufacturing: The simulation aspect. páginas 2085–2098, 12 2019. doi:10.1109/WSC40007.2019.9004659.

[34] Young Jun Son e Richard A Wysk. Automatic simulation model generation for simulation-based, real-time shop floor control. *Computers in Industry*, 45(3):291–308, 2001. URL: https://www.sciencedirect.com/science/article/pii/S0166361501000860, doi:https://doi.org/10.1016/S0166-3615(01)00086-0.