UNIVERSIDADE FEDERAL DO AMAZONAS
FACULDADE DE TECNOLOGIA
DEPARTAMENTO DE ELETRÔNICA E COMPUTAÇÃO

Fábio Ricardo Rocha dos Santos

Descrição e visualização 3D de processo de manufatura utilizando arquitetura baseada em Sistema Multiagente

Manaus

2023

Fábio Ricardo Rocha dos Santos

Descrição e visualização 3D de processo de manufatura utilizando arquitetura baseada em Sistema Multiagente

Monografia a ser apresentada à Coordenação do Curso de Engenharia Elétrica - Eletrônica da Universidade Federal do Amazonas, como parte dos requisitos necessários para obtenção do título de Engenheiro Eletricista com ênfase em Eletrônica.

Orientador: Prof. MSc. Rafael da Silva Mendonça

Universidade Federal do Amazonas
Faculdade de Tecnologia
Departamento de Eletrônica e Computação

Manaus
2023

Ficha Catalográfica

Ficha catalográfica elaborada automaticamente de acordo com os dados fornecidos pelo(a) autor(a).

Santos, Fabio Ricardo Rocha dos
S237d Descrição e visualização 3D de processo de manufatura utilizando arquitetura baseada em Sistema Multiagente / Fabio Ricardo Rocha dos Santos . 2023
78 f.: il. color; 31 cm.

Orientador: Rafael da Silva Mendonça
TCC de Graduação (Engenharia Elétrica - Eletrônica) - Universidade Federal do Amazonas.

1. Visualização. 2. Sistema Multiagente. 3. Sistema Cyber-físico. 4. Manufatura. I. Mendonça, Rafael da Silva. II. Universidade Federal do Amazonas III. Título

Fábio Ricardo Rocha dos Santos

Descrição e visualização 3D de processo de manufatura utilizando arquitetura baseada em Sistema Multiagente

Monografia a ser apresentada à Coordenação do Curso de Engenharia Elétrica - Eletrônica da Universidade Federal do Amazonas, como parte dos requisitos necessários para obtenção do título de Engenheiro Eletricista com ênfase em Eletrônica.

Trabalho aprovado. Manaus, 29 de junho de 2023:

Prof. MSc. Rafael da Silva Mendonça
Orientador

Prof. Dr. André Luiz Duarte Cavalcante
Membro Avaliador

Prof. Dr. Renan Landau Paiva de Medeiros
Membro Avaliador

Manaus
2023

Agradecimentos

Acima de tudo, aos meus pais, Leucivania Rodrigues Rocha e Silvio Junior Sousa dos Santos, pelos inúmeros sacrifícios feitos, não só durante a graduação, mas em toda a minha formação. Em especial, à minha mãe, pelo apoio incondicional desde sempre.

Aos meus irmãos e ao meu tio, Roni, por ter me recebido de braços abertos em Manaus e pelo imenso suporte durantes todos esses anos.

A todos os amigos e colegas que fiz durante a graduação, que me ensinaram muito e fizeram dessa jornada uma fase ainda mais memorável.

Aos excelentes professores com os quais tive a honra de ter aprendido. Em especial ao professor André Cavalcante, pela oportunidade de iniciação científica e por ter apresentado uma perspectiva profissional muito importante para minha carreira. Agradeço também ao professor Rafael Mendonça, pela paciência, conhecimento e orientação durante o PIBIC, publicação e TCC.

Resumo

Diversos paradigmas de manufatura têm sido propostos para lidar com a necessidade frequente de reconfiguração de linhas de montagem, especialmente para atender a demanda de customização em massa. Entre esses paradigmas, os sistemas evolutivos de produção (EAS/EPS) se destacam por sua capacidade de auto-organização no ambiente industrial. A arquitetura EPSCore, baseada em EAS/EPS e sistemas multiagentes, oferece flexibilidade e agilidade ao processo produtivo. No entanto, ela não contempla inicialmente recursos de visualização 3D e integração com CLP (Controlador Lógico Programável), que são importantes para o processo de simulação, que, por sua vez, é uma ferramenta essencial para o projeto de sistemas complexos, como os da indústria. Este trabalho propõe uma forma de integração da arquitetura EPSCore com um sistema de visualização 3D e um simulador de CLP, visando o desenvolvimento e visualização de processos de manufatura descritos por sistemas multiagentes. A proposta apresenta uma arquitetura para alcançar essa integração e, como validação, uma descrição de um estudo de caso.

Palavras-chave: Visualização. Sistema Multiagente. Sistema Cyber-físico. Manufatura.

Abstract

Several manufacturing paradigms have been proposed to deal with the frequent need for reconfiguration of assembly lines, especially to meet the demand for mass customization. Among these paradigms, evolvable production systems (EAS/EPS) stand out for their ability to self-organize in the industrial environment. EPSCore architecture, based on EAS/EPS and multi-agent systems, offers flexibility and agility to the production process. However, it does not originally contemplate 3D visualization and PLC (Programmable Logic Controller) integration resources, which are important for the simulation process, which in turn is an essential tool for the design of complex systems, such as those in industry. This work proposes a way to integrate EPSCore architecture with 3D visualization systems and PLC simulation, aiming the development and visualization of manufacturing processes described by multi-agent systems. The proposal presents an architecture to achieve this integration and, as validation, implementation of a case study.

Keywords: Visualization. Multiagent System. Cyber Physical System. Manufacturing.

Lista de ilustrações

Lista de abreviaturas e siglas

ACL	Agent Communication Language specifications - Especificações de Linguagem de Comunicação de Agente
AMS	Agent Management System - Sistema de Gerenciamento de Agente
CFP	Contract Net Interaction Protocol - Protocolo de Interação de Rede de Contrato
CLP	Controlador Lógico Programável
CPS	Cyber Physical System - Sistema Cyber-físico
DF	Directory Facilitator - Facilitator de Diretório
EAS	Evolvable Assembly System - Sistema de Montagem Evolutivo
EPS	Evolvable Production System - Sistema de Produção Evolutivo
FBD	Function Block Diagram - Diagrama de Bloco de Função
FIPA	Foundation for Intelligent Physical Agents - Fundação para Agentes Físicos Inteligentes
IDE	Integrated Development Environment - Ambiente de Desenvolvimento Integrado
IEC	International Electrotechnical Commission - Comissão Eletrotécnica Internacional
I/O	Input/Output - Entrada/Saída
IP	Internet Protocol - Protocolo de Internet
JADE	Java Agent DEvelopment Framework - Framework de Desenvolvimento de Agente Java
MAS	Multi-agent System - Sistema multiagente
MRA	Mechatronic Resource Agent - Agente Mecatrônico de Recurso
PDU	Protocol Data Unit - Unidade de Dado de Protocolo
POU	Program Organization Unit - Unidade Organizacional de Programa
RMA	Remote Management Agent - Agente de Gerenciamento Remoto

TCP	Transmission Control Protocol - Protocolo de Controle de Transmissão
YPA	Yellow Pages Agent - Agente de Páginas Amarelas

Sumário

1 Introdução

Dentre os desafios com os quais os sistemas de produção modernos têm de lidar, encontra-se a necessidade da criação de sistemas de montagem capazes de realizar a autoadaptação, a auto-organização, a auto-otimização, a autocorreção, a autoproteção e o autoconhecimento (CAVALCANTE, 2012). O cenário ideal, e portanto desejado, é que o tempo de reconfiguração do sistema de produção, para que o mesmo esteja apto a um novo produto, seja próximo a zero, aumentando-se a produtividade do sistema de montagem.

Assim, surgem propostas de arquiteturas e paradigmas da manufatura que propõem soluções para a fabricação ágil de produtos que devem ter alta variabilidade dentro de pequenos lotes de produção, a chamada customização em massa. Dentre esses paradigmas, destaca-se o EPS (Evolvable Production System), o qual propõe um sistema evolutivo baseado em módulos reconfiguráveis com tarefas específicas (ALSTERMAN; ONORI, 2004), permitindo a produção focada na agilidade de produção em meio à modificação dos módulos (meio de produção), em especial devido às modificações de produto.

Aos longo dos anos, o uso de agentes inteligentes e sistemas multiagentes (MAS) em aplicações industriais têm sido amplamente pesquisados e advocados como uma promessa realística para os problemas levantados (isto é, a necessidade de um sistema ágil capaz de evoluir e se adaptar às adversidades do mercado), conceituando o termo Agente Industrial (KARNOUSKOS et al., 2020). Paralelo a isso, nos últimos anos tem-se diversas implementações relevantes no âmbito da indústria 4.0 (SAKURADA; LEITÃO, 2020).

Um sistema multiagente é um grupo de agentes de autônomos (i.e. possui controle do seu estado interno, objetivo e comportamento, de forma a ser unicamente quem o controla) ou semiautônomos resolvedores de problemas específicos que juntam forças e trabalham cooperativamente para o seu próprio objetivo ou o objetivo da sua sociedade ou subsociedade, contemplando, portanto, um sistema dinâmico, flexível, robusto e escalável (LEITÃO; KARNOUSKOS, 2015).

Logo, tal tipo sistema se encaixa como implementação de EPS e são utilizados (CAVALCANTE, 2012), podendo atender a essas necessidades porque são entidades inerentemente modulares. Assim, cada agente pode assumir o controle de um recurso industrial, alinhando seus limites de atuação cibernética com os do sistema físico (KARNOUSKOS et al., 2020), explorando também o conceito de sistemas cyber-físicos (CPS) no contexto de transformação digital na indústria.

O presente trabalho traz uma descrição desse tipo de sistema, isto é, um sistema multiagente capaz de auto-organização, e que explora o conceito de agente cyber-físico, aplicado a um processo industrial. Logo, cada módulo do sistema industrial é modelado por um ou mais agentes e o processo final do sistema é alcançado com micro interações entre os agentes, que

refletem em macro comportamentos finais que implementam a produção. Para tal, foi utilizado a EPSCore, que encapsula e facilita a implementação de um EPS com MAS (MENDONÇA, 2016). Entretanto, o foco do trabalho é mostrar uma forma de integração desses sistemas com a visualização 3D, permitindo, assim, que o desenvolvedor consiga ver seu MAS industrial interagindo com o hardware industrial virtual 3D durante uma simulação.

1.1 Motivação e justificativa

A simulação permite ao projetista testar se as especificações do projeto são alcançadas, utilizando um ambiente virtual em vez de um físico, diminuindo significantemente o ciclo de *design*, reduzindo o custo de um projeto, e providenciando um feedback imediato do comportamento do sistema, o que tende a prover uma exploração mais compreensiva das alternativas de *design* do projeto, logo, um melhor *design* (SINHA et al., 2001). Isso é especialmente verdadeiro aplicando à manufatura, onde os sistemas podem ser enormes, complexos, apresentar altos riscos, e a prototipação e validação física se torna uma tarefa árdua e por vezes cara (HOSSEINPOUR; HAJIHOSSEINI, 2009). Tratando de sistemas flexíveis e evolutivos, os quais inerentemente buscam contemplar variadas possibilidades, a simulação e visualização se torna um aliado quase obrigatório.

A arquitetura EPSCore é uma modelagem baseada em EPS capaz de reconhecer os módulos que estão ativos em um determinado momento e se auto-organizar, de modo a formar a sociedade de agentes necessária para a execução de algum objetivo no sistema (MENDONÇA, 2016). Ela possui, portanto, a característica chamada *plug and produce*, em que cada módulo do sistema pode ser retirado do, ou colocado no, sistema como parte integrante de seu funcionamento normal, permitindo, assim, flexibilidade e agilidade à produção, o que, por sua vez, pode prover uma vantagem competitiva (SUZIĆ et al., 2018).

Logo, a arquitetura EPSCore trata-se de uma modelagem baseada em sistema multiagente e EPS, onde cada módulo físico do sistema (assim como seus serviços e capacidades) é modelado por um ou mais agente. A arquitetura define uma estrutura básica de organização dos agentes por camadas e disponibiliza algumas classes base em Java para uso. Entretanto, a mesma e suas classes se limitam ao sistema multiagente Java, não contemplando originalmente, portanto, uma implementação intrínseca para a visualização 3D, assim como integrações com CLP, que é uma interface controladora comum amplamente utilizada em sistemas de produção.

Portanto, este trabalho utiliza a arquitetura EPSCore como apoio central na descrição de um processo de manufatura (estudo de caso) utilizando um sistema multiagente, mas busca estender o seu alcance com modificações e especificações em código, de forma a desenvolver uma integração com um sistema de visualização 3D de processo industrial e com um simulador e ambiente de desenvolvimento de CLP gratuito, o qual conta com diversas extensões de soluções em automação industrial. A integração é feita de tal forma que o sistema Java é conectado

ao ambiente de desenvolvimento de CLP, que por sua vez se conecta com um software de simulação 3D, capaz de simular plantas industriais. Logo, o presente trabalho tem como principal contribuição uma proposta de arquitetura para visualização e desenvolvimento de processos de manufatura em 3D, baseados em sistemas multiagentes e EPS, utilizando a arquitetura EPSCore.

1.2 Objetivos

1.2.1 Objetivos gerais

É de objetivo geral desenvolver uma forma de conectar um sistema multiagente na plataforma EPSCore com uma planta 3D e CLP simulados. Além disso, deseja-se validar tal solução com uma demonstração virtual 3D de um sistema multiagente aplicado à manufatura (planta simulada, CLP e sistema multiagente).

1.2.2 Objetivos específicos

Em particular, para alcançar o objetivo geral, será necessário:

- implementar integração da plataforma EPSCore com um CLP simulado;
- implementar comunicação entre CLP e software de visualização 3D;
- realizar experimentos de cenários virtuais 3D integrados à arquitetura EPSCore;
- realizar a descrição de um processo de manufatura utilizando a arquitetura EPSCore, CLP e Planta 3D;

2 Fundamentação Teórica

2.1 Agentes

A palavra agente é, intuitiva e linguisticamente, definida como aquele que age. Apesar de não haver uma só definição, na computação, é um consenso que um agente é uma entidade computacional que possui autonomia, características de um sistema arbitrário e/ou se comporta como um agente humano, trabalhando para alguns clientes em busca de sua própria agenda (BELLIFEMINE; CAIRE; GREENWOOD, 2007).

Assim, um agente pode ser definido como uma entidade de software que pode perceber e/ou atuar em seu ambiente (por uso de sensores e atuadores). Um agente racional ou inteligente é, portanto, um agente capaz de tomar uma decisão sobre como atuar, a partir do que percebe do seu meio.

Para esse trabalho, outro conceito muito importante a ser estabelecido é o de agente cyber-físico, mas para isso, conceitos como o de módulo e entidade cyber-física precisam ser esclarecidos. Um módulo é uma entidade autocontida que possui uma função específica, uma interface bem definida e a possibilidade de interagir com outros módulos (CAVALCANTE, 2012). Já um uma entidade cyber-física é uma parte de um sistema cyber-físico, ou CPS, que, por sua vez, é definido como a integração do processo físico e de cálculo, que inclui processo físico de controle e monitoramento e computadorizado em rede (LEE, 2007). Sendo assim, uma entidade cyber-física é aquela que integra seu hardware, permitindo esse funcionar com uma representação cibernética, atuando como uma representação virtual da parte física (LEITÃO et al., 2016).

Por fim, um agente cyber-físico pode ser definido como um agente que integra o módulo ao sistema (rede), podendo receber e enviar informações ao hardware.

É importante salientar que um mesmo módulo pode estar associado a mais de um agente. Em uma abordagem multiagente, vários agentes racionais (software) podem ser criados para controlar um módulo e do ponto de vista do sistema maior a que este módulo será conectado, este subsistema multiagente seria encarado apenas como um agente (CAVALCANTE, 2012), exaltando a hierarquia.

Quando um conjunto de agentes do mesmo tipo ou de tipos diferentes são implementados de forma que, conjuntamente, resolvam um problema, temos um sistema Multiagentes (Multi-agent Systems - MAS). Sendo assim, um MAS pode ser definido como sistema que compreende dois ou mais agentes ou agentes inteligentes (MCARTHUR et al., 2007).

2.1.1 Comunicação entre agentes: Protocolos FIPA

A fim de permitir a interoperabilidade entre agentes físicos, independente da linguagem de programação utilizada para o seu desenvolvimento, a Foundation for Intelligent Physical Agents (FIPA) definiu vários protocolos e regras para comunicação entre agentes em vários níveis: aplicação, arquitetura, comunicação, gerenciamento de agentes e transporte de mensagens. Dentre esses, é especialmente importante, para esse trabalho, citar a Agent Communication Language specifications (ACL) e o FIPA Contract Net Interaction Protocol.

As especificações FIPA ACL representam um conjunto de normas que se destinam a padronização da comunicação propriamente dita dos agentes. Uma mensagem FIPA ACL contém um conjunto de um ou mais parâmetros de mensagens, como destinatário, performativo e diversos outros parâmetros associados às propriedades que auxiliam na comunicação. Contudo, nem sempre as aplicações necessitam de todos. O único parâmetro obrigatório em todas as mensagens ACL é o `performative`.

O FIPA Contract Net Interaction Protocol é usado na comunicação entre agentes do tipo $1 \times n$ em que um agente demonstra interesse em certo serviço mandando uma mensagem (Call For Propose - `CFP`) para os agentes que o oferecem. Assim, tais agentes podem retornar mensagens (`Propose`) com suas propostas ou simplesmente recusar (`REFUSE`). Em seguida, o agente iniciador pode escolher com quais agentes a solicitação do serviço será feita.

2.1.2 JADE – Java Agent DEvelopment

Java Agent DEvelopment Framework (JADE) é um framework para o desenvolvimento de aplicações multiagentes em Java seguindo as especificações da FIPA. Ou seja, é uma plataforma genérica para desenvolvimento de sistemas multiagente. O framework é distribuído na forma de um pacote Java, de forma que sua distribuição e uso em aplicações específicas de variados contextos é simplificada (CAVALCANTE, 2012). A escolha do framework para o projeto é justificada pelo fato de a Arquitetura EPSCore ter sido desenvolvida utilizando o mesmo.

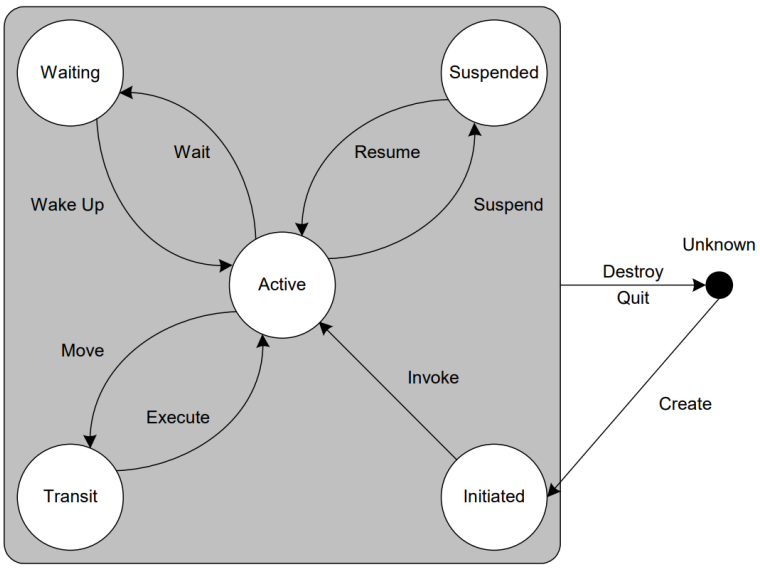
Ao iniciar o ambiente JADE são criados automaticamente alguns agentes especiais, são eles: Agent Management System (AMS), Directory Facilitator (DF) e o Remote Managment Agent (RMA) (opcional). O AMS é único e é por responsável controlar o acesso e uso da plataforma, registrando todos os agentes. Já o DF é um agente de suporte que realiza o serviço de páginas amarelas, ou seja, serve de ponte entre os serviços que são solicitados/aceitos organizando cada agente com suas skills numa lista. Por fim, o RMA é um agente responsável por um gerenciamento da plataforma, seus contêineres e dos demais agentes.

2.1.3 Ciclo de vida de um agente

No JADE, um agente é modelado através de uma classe Java, a classe `Agent`, a qual implementa uma *thread* Java que executa comportamentos continuamente. O ciclo de vida de um

agente é definido como uma máquina de estados: após ser iniciado, o agente vai para o estado ativo e ali permanece até ser explicitamente colocado em espera ou suspenso (bloqueado). No estado de trânsito, o agente se encontra em trânsito entre dois contêineres. A Figura 1 ilustra o ciclo.

Figura 1 – Ciclo de vida de um agente.



Fonte: (CAIRE, 2009).

2.1.4 Comportamentos

Durante o tempo de vida do agente, esse irá executar seus comportamentos. Um comportamento é uma entidade de execução dentro do agente, modelada através da classe `Behaviour`. Um agente pode ter vários comportamentos e esses são normalmente adicionados ao agente no método `setup()`, mas podem ser adicionados de qualquer lugar do agente, inclusive a partir de outros comportamentos.

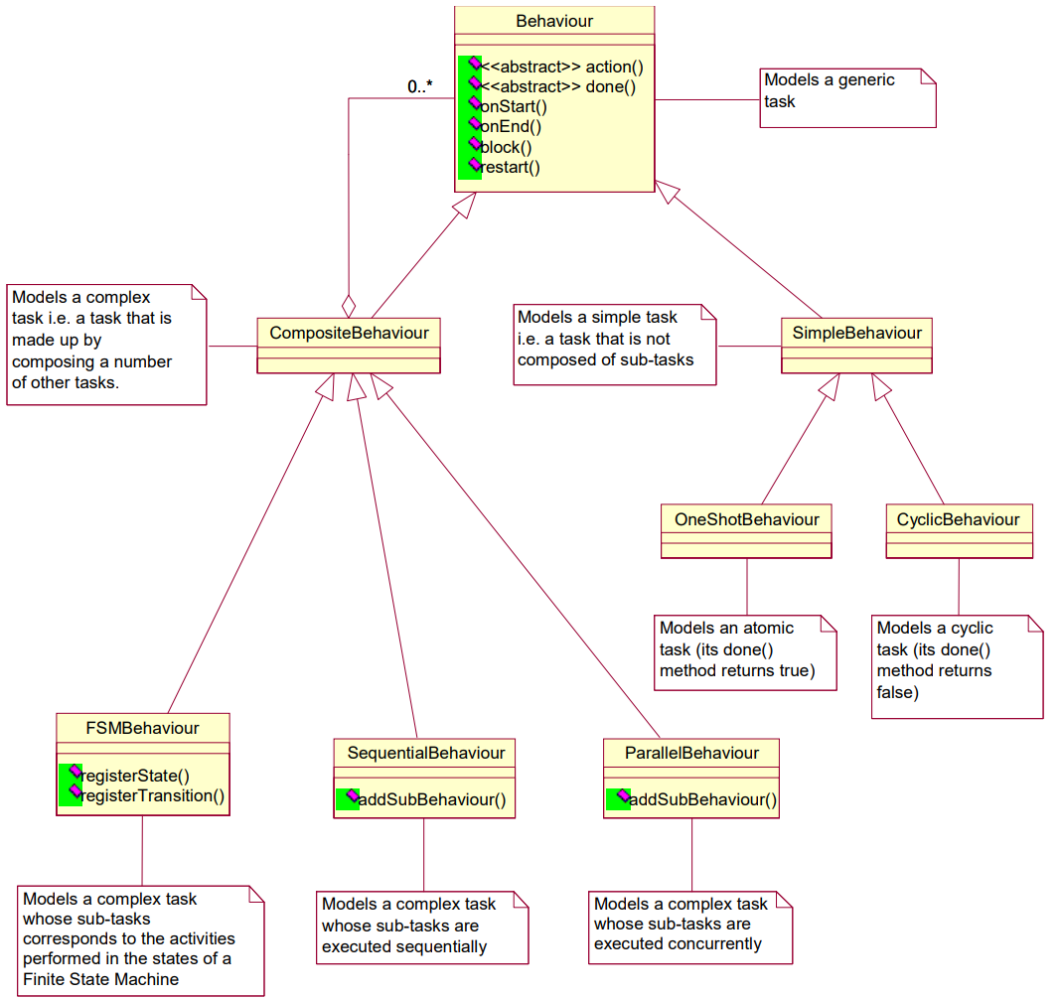
Em um comportamento é possível definir uma função específica, como a captura de dados externos, análise da situação corrente, definição da ação no meio ou a ação propriamente dita (CAVALCANTE, 2012). O comportamento global do agente, isto é, o comportamento resultante final, será o resultado das ações de todos os comportamentos do agente. O JADE define alguns comportamentos padrões que estão disponíveis e têm funcionalidades próprias:

- CyclicBehaviour: é a implementação da classe `SimpleBehaviour` com o método `done()` sempre retornando `False` fazendo com que o comportamento seja cíclico;
- OneShotBehaviour: permite ao programador implementar comportamentos que serão executados somente uma vez;

- SequentialBehaviour: agenda no JADE a execução de subcomportamentos de modo sequencial e termina quando todos são executados;
- ParallelBehaviour: agenda no Jade a execução de subcomportamentos de modo paralelo e pode terminar depois que todos ou de uma determinada quantidade de comportamentos terminarem;
- FSMBehaviour: agenda no JADE a execução de subcomportamentos como uma máquina de estados em que os eventos são os valores de retorno do método `onEnd()` e cada subcomportamento é um estado da máquina.

A Figura 2 mostra a hierarquia das classes dos comportamentos através do diagrama de classes. Como é possível observar, tem-se duas ramificações principais: comportamentos simples (`CyclicBehaviour` , `OneShotBehaviour`) e compostos (`SequentialBehaviour` , `ParallelBehaviour` , `FSMBehaviour`).

Figura 2 – Diagrama de classe dos comportamentos no Jade.



Fonte: (CAIRE, 2009).

2.2 Auto-organização e Emergência

Apesar do senso comum não sugerir, emergência e auto-organização referem-se a dois fenômenos completamente distintos e a confusão deve ser evitada usando cada conceito corretamente e não como sinônimos. Por isso, essa seção tem por objetivo esclarecer tais conceitos e, para isso, utiliza o trabalho de Wolf e Holvoet (2004).

O conceito de auto-organização é bem intuitivo: um processo dinâmico e adaptável, em que os sistemas adquirirem e mantêm estruturas por eles mesmos (sem controle externo), onde essa estrutura pode ser espacial, temporal ou funcional. Aplicado a sistemas de manufatura, para esse trabalho, destaca-se a organização automática, sem controle externo, do sistema de produção ao ser modificado (inclusão ou remoção de um módulo), para uma pronta produção.

Por outro lado, um sistema exibe emergência quando há emergentes coerentes no nível macro que dinamicamente aparecem das interações das partes no nível micro. Tais emergentes são uma novidade radical com respeito às partes individuais do sistema. Em outras palavras, emergência é a característica que permite um comportamento global (do sistema), não antes programado explicitamente, a partir de comportamentos/fenômenos individuais (partes que constituem o sistema). Exemplos simples de emergência são: caminhos globais de feromônios que surgem dos caminhos locais desses feromônios, o movimento de enxame de um bando de pássaros, um engarrafamento devido às interações dos carros e etc.

Portanto, é evidente que um sistema pode tanto apresentar emergência sem auto-organização, como auto-organização sem emergência, uma vez que os conceitos são distintos e separáveis. Além disso, o sistema pode apresentar as duas características simultaneamente, formando um sistema altamente robusto em relação à flexibilidade e à adaptabilidade.

2.3 Sistemas evolutivos: EPS/EAS

Os termos Evolvable Assembly System (EAS) e Evolvable Production Systems (EPS) foram originalmente usados por ONORI (2002); ALSTERMAN; ONORI (2005), em suas abordagens aos sistemas de montagem e ao sistemas produtivos, respectivamente. EPS e EAS são conceitos diferentes, porém, muito similares. Antes de diferenciá-los, é necessário entender o que é um sistema evolutivo.

Um sistema evolutivo pode ser definido como um sistema dinâmico e auto-organizado, que consegue modificar a sua estrutura devido às mudanças ambientais relevantes. Como dito na seção anterior, uma mudança especialmente relevante é a inserção, remoção ou reposição de módulos ao sistema, o que caracteriza a plugabilidade, a qual é parte do funcionamento normal do sistema. Esse conceito concorda com a definição para EAS/EPS de (ONORI; BARATA; FREI, 2006).

Ainda para os autores, “EPS é baseado em muitos elementos (módulos do sistema)

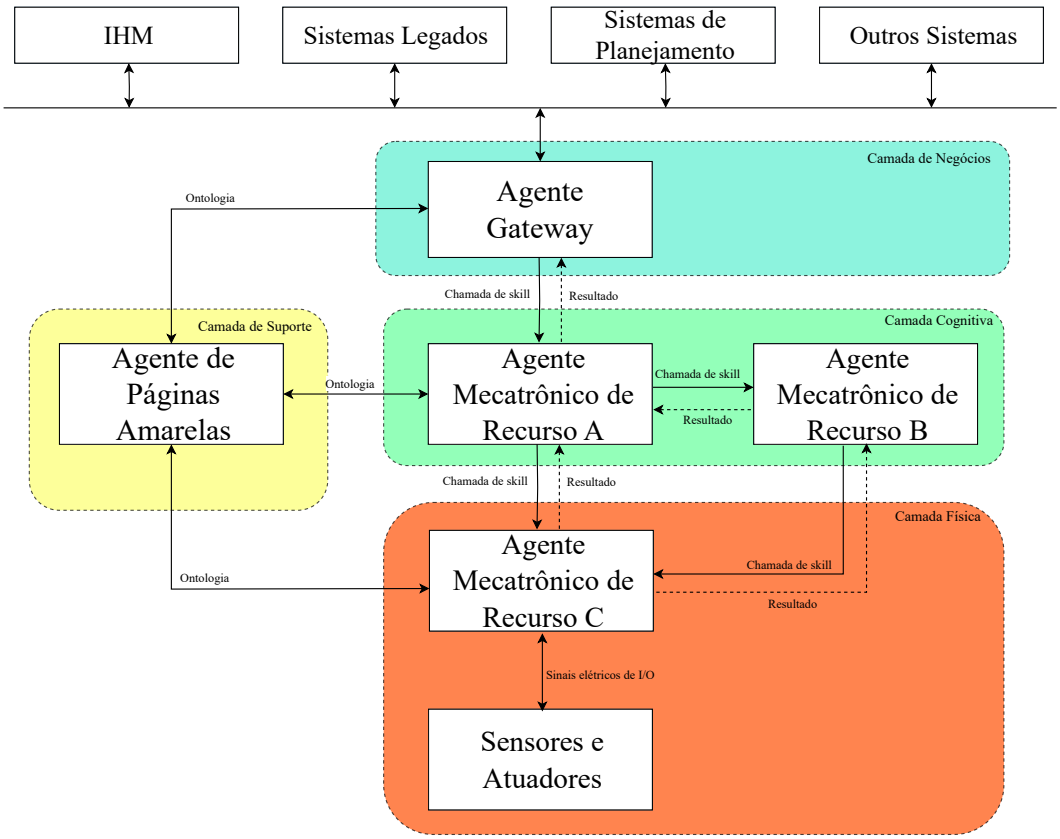
simples, específicos à tarefa e reconfiguráveis, que permitem evolução contínua do sistema de montagem". Já EAS é "um sistema de montagem que pode coevoluir conjuntamente com o produto e processo de montagem" (FREI et al., 2009).

Em outras palavras, EAS e EPS são conceitos similares, exceto pelo nível que são considerados, pois EAS está focado no nível do dispositivo, enquanto EPS está focado no nível da fábrica. Portanto, no nível da célula, dependendo das circunstâncias, um mesmo sistema pode ser chamado tanto de EAS quanto de EPS (CAVALCANTE, 2012). Por isso, nesse trabalho, os termos serão usados como sinônimos.

2.4 Arquitetura EPSCore

A arquitetura EPSCore foi inicialmente proposta por (MENDONÇA; CAVALCANTE; JUNIOR, 2016), posteriormente descrita em (MENDONÇA, 2016) e é inspirada na arquitetura IADE, que é a arquitetura de um projeto da União Européia. A arquitetura EPSCore é fundamentada basicamente em MAS e EPS, de forma que a inteligência do sistema está distribuído entre os agentes, isto é, a confecção de novos produtos, pelo sistema produtivo, acontece por meio das interações entre os agentes do sistema.

Figura 3 – Arquitetura EPSCore.



Fonte: Adaptada de (MENDONÇA, 2016).

A divisão em camadas existe como abordagem simplificada para facilitar a solução de problemas mais complexos partindo do princípio de "dividir para conquistar". De acordo com (MENDONÇA, 2016), são elas:

- Camada física: é formada por agentes cyber-físicos e tem a responsabilidade de perceber e de agir sobre o mundo físico através do acionamento de atuadores e sensores presentes no sistema;
- Camada cognitiva: é formada por agentes cyber-físicos que contêm inteligência capaz de coordenar outros agentes cyber-físicos através da troca de mensagens e chamadas de skills que acontecem no sistema;
- Camada de suporte: é formada por agentes não cyber-físicos que dão suporte à execução de serviços e descoberta de funcionalidades no sistema para os demais agentes;
- Camada de negócios: contém agentes não cyber-físicos que representam a ponte de comunicação entre a arquitetura EPSCore e os outros sistemas que podem ser desde um monitor a um outro sistema cyber-físico.

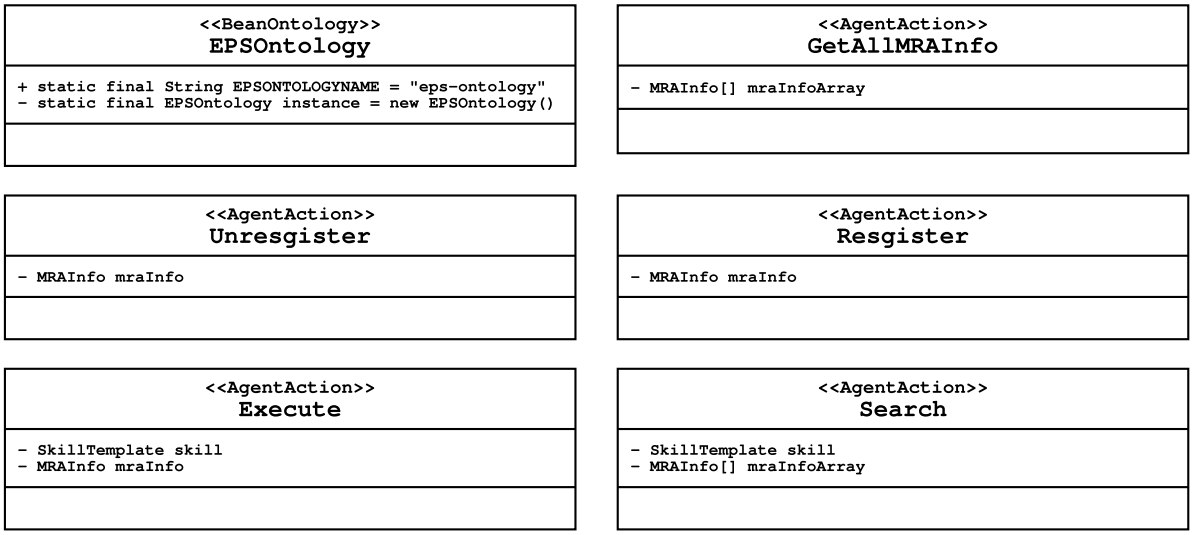
2.4.1 Implementação

Como dito anteriormente, a implementação da arquitetura é feita em Java, com o framework JADE. Para conseguir uma comunicação eficaz e concisa, a arquitetura conta com classes que estabelecem conceitos dentro de uma ontologia própria da arquitetura. A ontologia define uma estrutura comum de significados para serem usados pelos agentes na comunicação sem que aconteça a possibilidade de ambiguidades. Assim, é criado um vocabulário semântico para a troca de mensagens da aplicação. No que diz respeito à ontologia, segundo (MENDONÇA, 2016), temos as seguintes classes:

- EPSOntology: é um BeanOntology (instância da classe nativa do Java para ontologia) responsável pela definição, registro e instanciação da ontologia;
- Unregister: é um AgentAction (classe do JADE que representa uma ação de um agente) encarregado de cancelar o registro dos agentes cyber-físicos dentro do agente páginas amarelas;
- Execute: é um AgentAction responsável pela chamada e execução dos skills;
- Register: é responsável pelo registro dos agentes cyber-físicos no agente de páginas amarelas através de informações contidas no MRAInfo do mesmo;
- Search: disponibiliza uma busca às skills (funcionalidades) dos agentes registrados no sistema.

- GetAllMRAInfo: responsável por realizar uma solicitação de todos os MRAInfo cadastrados.

Figura 4 – Classes da ontologia na arquitetura EPSCore.

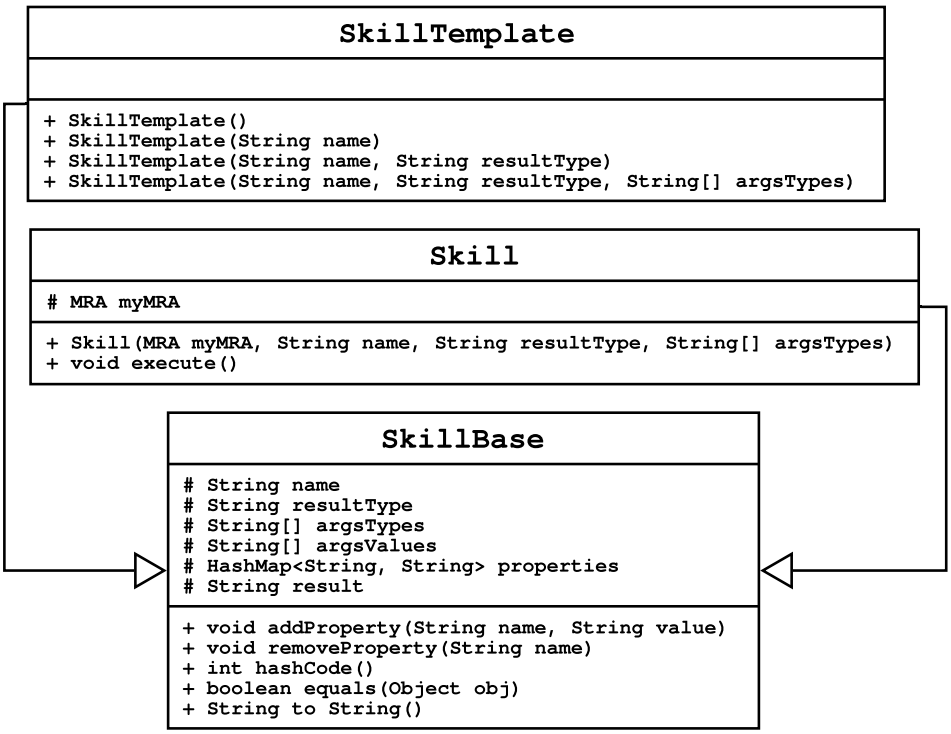


Fonte: (MENDONÇA, 2016).

Além disso, é importante também salientar o conceito de skill, que é muito importante para a arquitetura, pois permeia todo o funcionamento da mesma. Uma skill é um serviço de um agente, de forma que ele o implementa e oferece aos outros agentes. É importante notar que as funcionalidades da planta física são abstraídas e oferecidas como serviços utilizando o conceito de skill. A arquitetura conta com 3 classes para implememententar esse conceito, são elas:

- SkillBase: é definida como a classe principal que contém um padrão de informações referentes ao nome, tipo, argumentos, tipos de retorno e às propriedades das habilidades padronizadas;
- Skill: é usada para realizar a chamada de serviços dos agentes do sistema. Através dela é possível executar as skills dos agentes do sistema;
- SkillTemplate: é um conceito que define um conjunto de templates (padrões) para realizar a troca de informações de determinadas skills no sistema.

Figura 5 – Classes do conceito de Skill na arquitetura EPSCore.

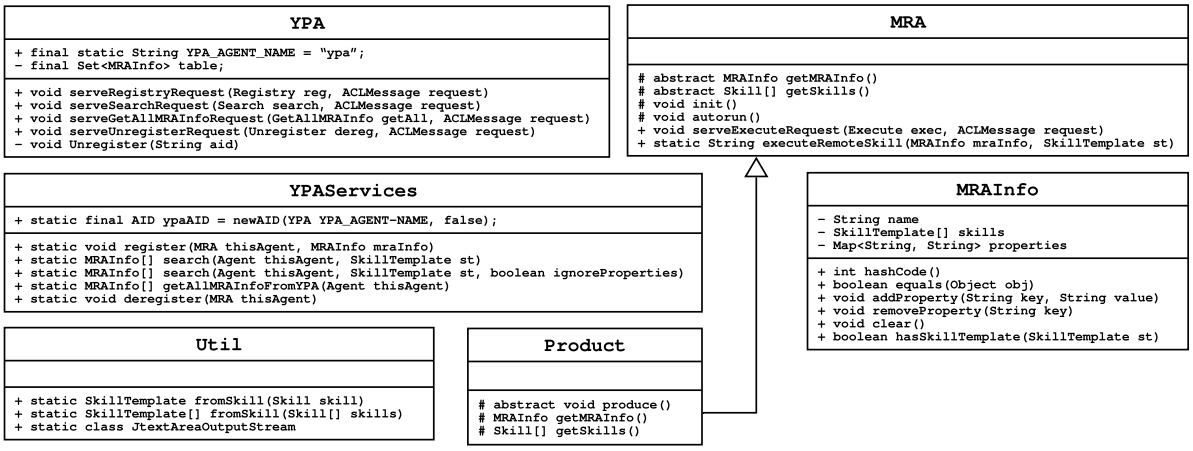


Fonte: (MENDONÇA, 2016).

Outras classes importantes, são:

- MRA: define um agente cyber-físico, ou seja, para se criar um agente cyber-físico é necessário que a classe em questão, seja filha de MRA;
- YPA: é a classe que define o agente de páginas amarelas. Interage com todas as outras camadas da arquitetura EPSCore e oferece serviços como: registro, pesquisa e cancelamento de registro;
- MRAInfo: são informações específicas sobre os agentes cyber-físicos;
- YPAServices: é chamado esse método para criar um agente de execução remota para o YPA;
- Product: é um produto genérico. Implementa o método `produce()` que ativa um plano de produção;
- Util: é a classe responsável pela mudança/conversão da skill. Ela realiza a mudança de SkillTemplate para Skill.

Figura 6 – Outras classes da arquitetura EPSCore.



Fonte: (MENDONÇA, 2016).

2.5 Protocolos de comunicação

2.5.1 TCP/IP

A suíte de protocolos TCP/IP é um conjunto de protocolos composto por cinco camadas: física, enlace, rede, transporte e aplicação. Tais protocolos são relativamente independentes e podem ser mesclados e combinados dependendo das necessidades do sistema (FOROUZAN, 2009). Dentro da camada de transporte, tem-se principalmente o TCP, que é um protocolo de fluxo confiável, com tratamento de erros, e que divide o fluxo de dados em unidades menores, denominadas segmentos. Tais pacotes são enviados com a garantia de ordem e integridade da informação.

2.5.2 Modbus TCP/IP

Modbus é um protocolo de mensagens que fornece comunicação cliente/servidor entre dispositivos conectados em diferentes tipos de barramentos ou redes. Ele é baseado em "solicitação e resposta"e oferece serviços especificados por códigos de função, que são elementos de PDUs (Protocol Data Unit - Unidade de dado de protocolo) de solicitação/resposta Modbus (Modbus Organization, 2012). As mensagens enviadas são de um cliente para os dispositivos servidores, e, na mensagem, o campo do código de função informa ao servidor que tipo de ação deve ser executada, enquanto o campo de dados contém informações adicionais que o servidor usa para executar a ação definida pelo código de função, como, por exemplo, a quantidade de itens a serem manuseados (Modbus Organization, 2012).

Modbus TCP/IP é uma variante da família Modbus, de comunicação simples e independente de fornecedor destinados à supervisão e controle de equipamentos de automação.

Especificamente, esse protocolo abrange o uso de mensagens Modbus usando os protocolos TCP/IP (SWALES et al., 1999). Para esse trabalho, o protocolo em questão foi utilizado na comunicação do sistema multiagente com o CLP simulado.

2.5.3 OPC UA

OPC UA é um padrão independente de plataforma através do qual vários tipos de sistemas e dispositivos podem se comunicar enviando mensagens de solicitação e resposta entre clientes e servidores ou mensagens de rede entre publicadores e assinantes em vários tipos de redes (OPC Foundation, 2023). Além disso, os servidores podem fornecer acesso a históricos de dados, bem como alarmes e eventos para notificar os clientes sobre alterações importantes. Tais dados podem ser codificados de várias maneiras para disponibilizar portabilidade e eficiência (OPC Foundation, 2023). Para esse trabalho, o protocolo em questão foi utilizado na comunicação da planta virtual 3D com o CLP simulado.

3 Proposta de Trabalho

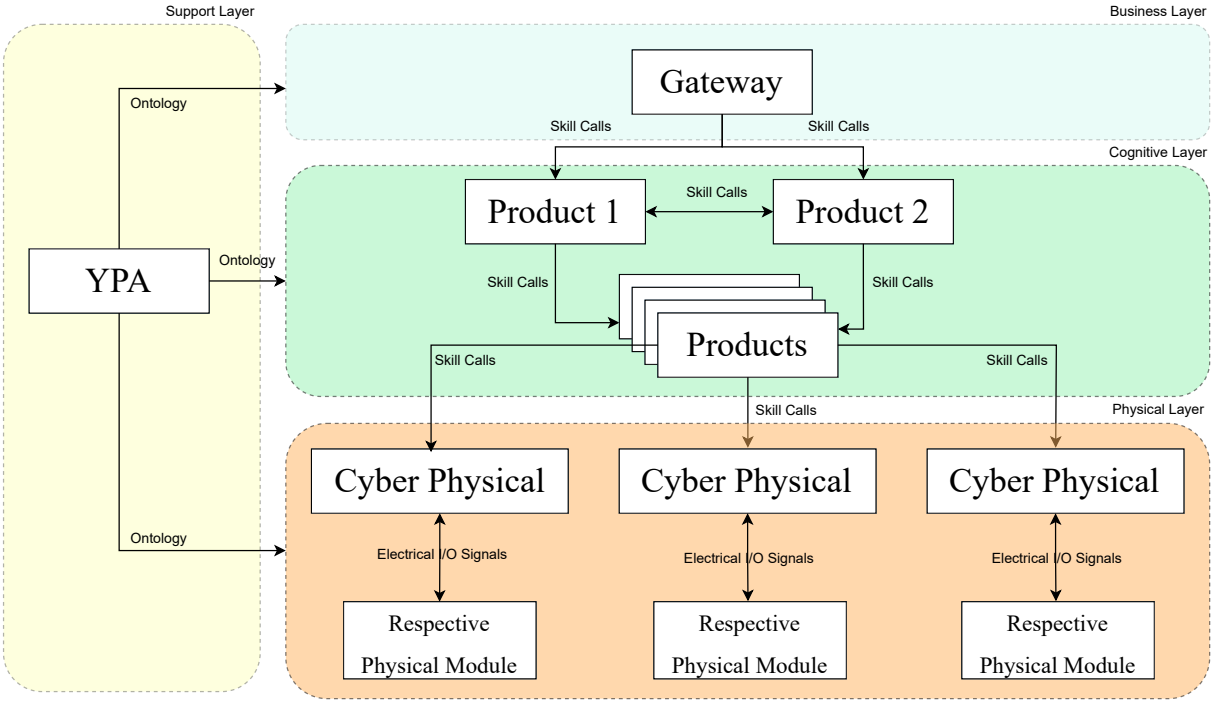
O presente trabalho tem como objetivo estabelecer uma forma de visualizar e simular processos de manufatura descritos por sistemas multiagentes que utilizam a arquitetura EPSCore. Logo, é proposto uma forma de conectar o sistema Java com uma planta virtual 3D do processo industrial, a qual tem suas funcionalidades modeladas via CLP, de modo a abstrair os serviços das mesmas para o sistema de mais alto nível (MAS).

Assim, os agentes que representam os módulos físicos (i.e. agentes cyber-físicos) podem chamar as funcionalidade através do interfaceamento provido pelo CLP, que encapsula a complexidade do serviço e os oferta através de variáveis de gatilho e confirmação.

3.1 Integração da arquitetura EPSCore

A Figura 7 mostra uma releitura da arquitetura EPSCore, deixando claro o papel do agente cyber-físico, que é representar e controlar um módulo físico. A Figura 8, por sua vez, mostra o conceito da integração da arquitetura com a simulação e visualização 3D do processo, através de um CLP virtual e um software de simulação de plantas industriais 3D.

Figura 7 – Arquitetura EPSCore atualizada.

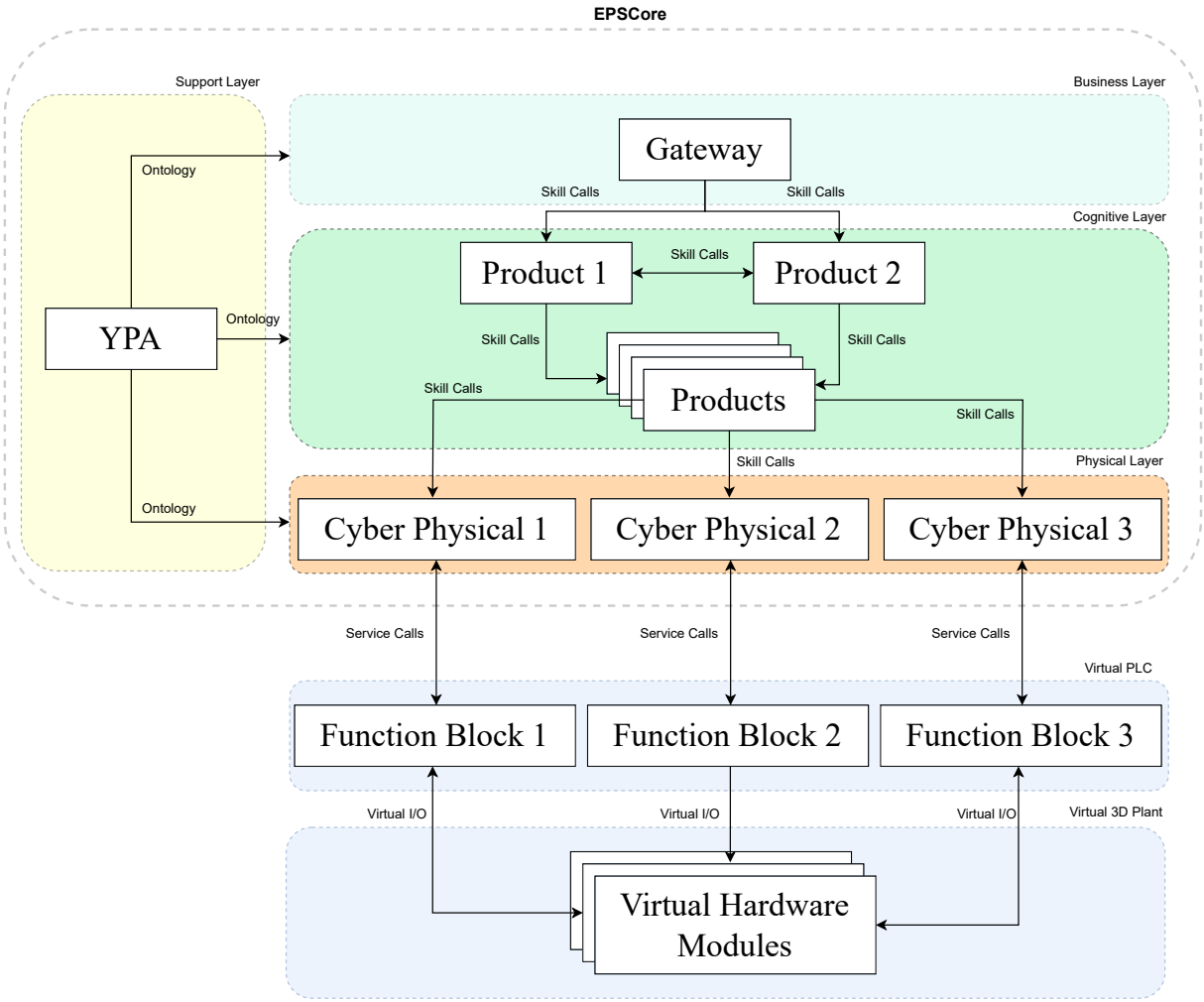


Fonte: De autoria própria.

Em suma, os serviços da planta virtual, a qual simula a planta física, são implementados

via CLP. Especificamente, instâncias de blocos funcionais dos módulos em questão. Esses blocos funcionais são desenhados de forma a realizar o encapsulamento do serviço para o sistema multiagente. Logo, para cada módulo da planta, existe um bloco funcional responsável por realizar a interface para o agente cyber-físico em questão.

Figura 8 – Integração proposta da arquitetura EPSCore.



Fonte: De autoria própria.

Portanto, este trabalho se limita aos seguintes pontos.

1. Realizar a conexão da planta 3D com o CLP, de forma que a mesma seja controlável via Ladder/FBD (Funcion Block Diagram).
2. Mostrar uma forma de abstrair as funcionalidades da planta via serviço implementado no CLP, de forma a ofertar para um MAS.
3. Estender o alcance do agente cyber-físico, modelado originalmente pela classe `MRA` na arquitetura, com uma forma de conectar a um CLP virtual (ou seja, escrever e ler variáveis do mesmo). Tal extensão implica em modificações em código da classe `MRA`.

4. Obter a implementação do serviço da planta virtual como skill no sistema multiagente, uma vez que as partes foram conectadas.

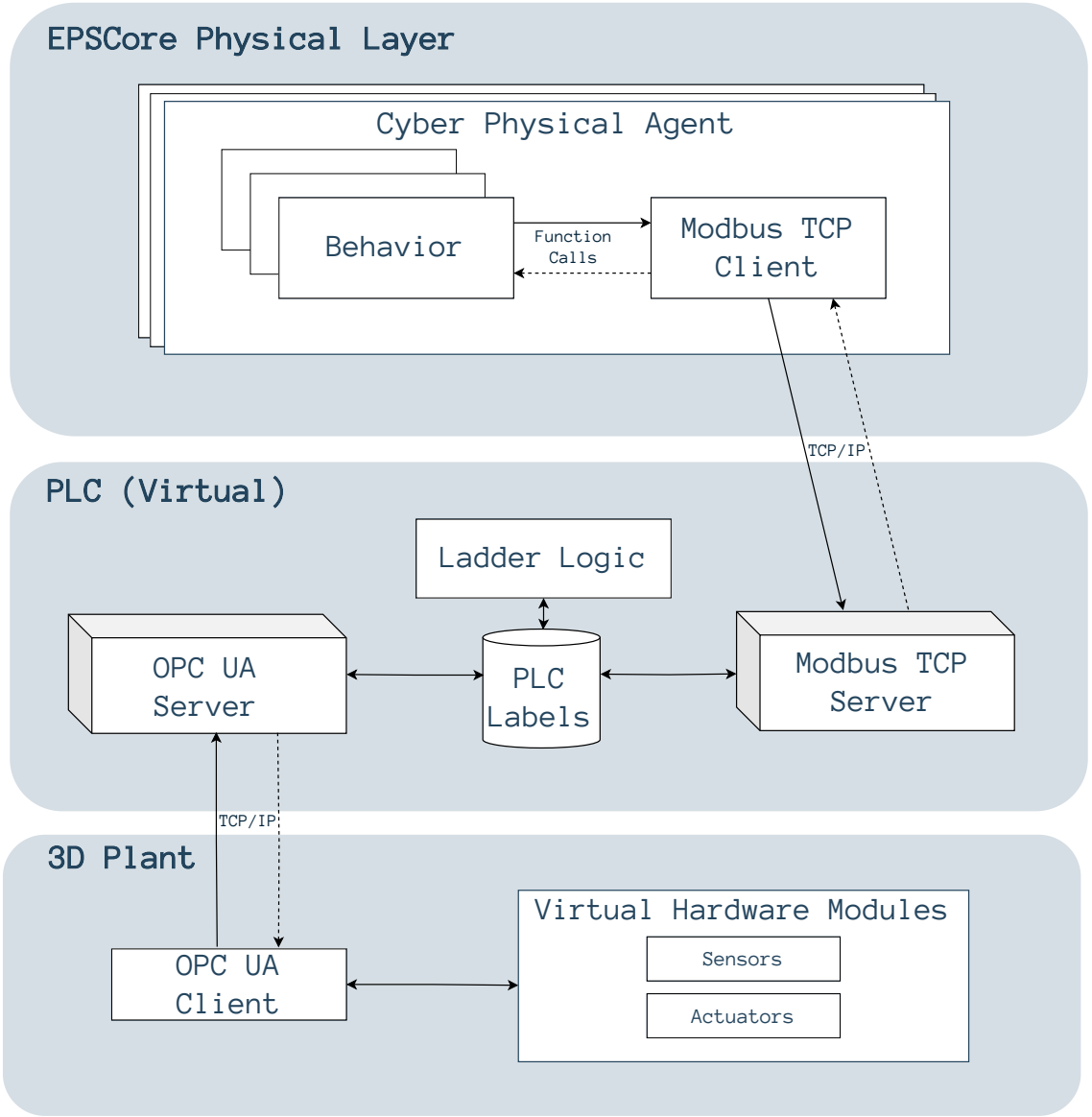
5. Utilizar dessa integração para simulação e visualização do MAS industrial.

Com isso, é possível testar e visualizar o sistema multiagente industrial ao longo do desenvolvimento, desde que seja feito a modelagem/abstrações das funcionalidades via CLP para que os agentes cyber-físicos possam interagir com a planta ao chamar suas skills. Além disso, a visualização/simulação pode ser usada para recurso didático no ensino de MAS voltados para processos de manufatura.

4 Implementação

Considerando a camada inicial da arquitetura EPSCore, aplicada no conceito de shop-floor, denominada camada física, que é povoada por agentes responsáveis por interagir com, e representar, módulos físicos mecatrônicos, foi estabelecido uma extensão e padronização de comunicação na mesma com um simulador de CLP e um simulador de planta de manufatura. Esta seção objetiva descrever os detalhes dessas especificações.

Figura 9 – Arquitetura proposta.



Fonte: De autoria própria.

A Figura 9 mostra uma visão geral da proposta de arquitetura para a extensão e conexão

da arquitetura EPSCore com o ambiente de desenvolvimento e simulação virtual para uma planta controlada por um CLP. Pode-se observar uma pilha de 3 camadas de tecnologias/ambientes: a mais acima é o sistema multiagente arquitetado tal qual as especificações EPSCore, feito em Java; a intermediária é o ambiente simulado de CLP com a IDE CODESYS; e, por fim, a camada física final, que é a planta virtual no Factory I/O.

No que diz respeito à primeira (camada física da arquitetura EPSCore), existe uma classe que modela um agente cyber-físico, o qual interage diretamente com um módulo físico, e foi originalmente nomeada de `MRA`. Foi adicionado na classe em questão um objeto cliente Modbus TCP/IP capaz de se comunicar com um servidor Modbus TCP/IP.

Já na camada do ambiente virtual de CLP, foi feita a programação em Ladder e FBD de cada módulo, assim como serviços ofertados pelos mesmos. Tal descrição manipula variáveis do CLP que estão amarradas aos servidores (implementados na IDE) que interagem com o sistema Java (servidor Modbus TCP/IP) e a planta 3D no Factory I/O (servidor OPC UA).

O servidor Modbus TCP/IP mapeia variáveis (labels) que representam a chamada ou gatilho de um serviço implementado em Ladder já arquitetado para a oferta a um sistema de software de mais alto nível. Também é mapeado o conjunto de confirmações de finalização de serviço e resultados desses serviços. O servidor OPC UA, por sua vez, mapeia as variáveis (labels) que representam as entradas e saídas (sensores e atuadores) da planta virtual no Factory I/O.

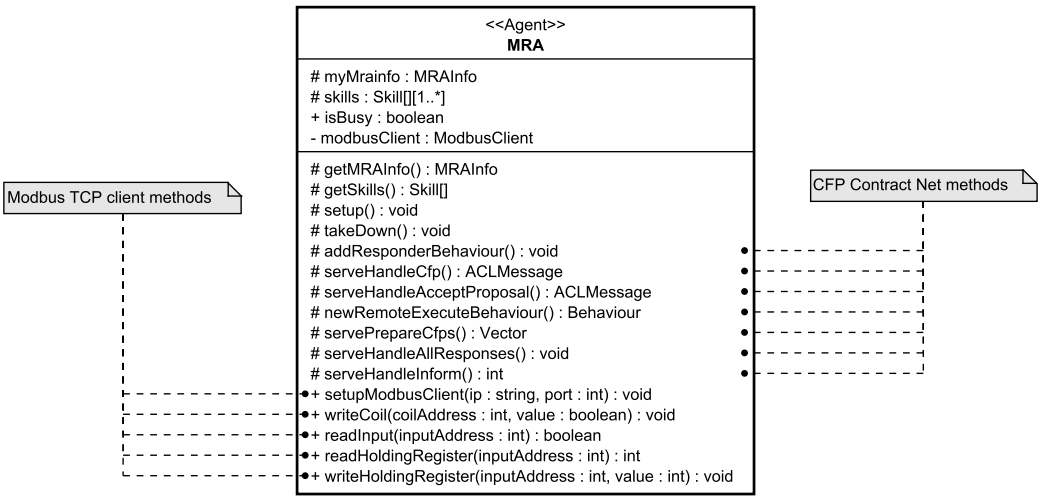
Por fim, na camada da planta virtual é utilizado um cliente OPC UA, implementado nativamente pelo software Factory I/O, para consultar e atualizar as variáveis do servidor OPC UA no CLP. Essas variáveis são então amarradas ao atuadores e sensores da planta. Assim, ao simular, os módulos físicos são controlados em tempo real pelo CLP.

4.1 Modificações na arquitetura EPSCore

4.1.1 Classe MRA

Dentre as modificações feitas, são especialmente importantes as feitas na classe `MRA`, a qual modela um agente cyber-físico. Essa importância se dá pelo fato de tal agente ser a conexão, na parte de software em alto nível, implementando os agentes de mais baixo nível, os quais executam e negociam as skills base para o sistema. Por isso, as seções seguintes têm como objetivo explanar duas grandes modificações/extensões feitas para permitir a integração proposta: a rede de contrato e o cliente Modbus TCP/IP.

Figura 10 – Classe MRA modificada.



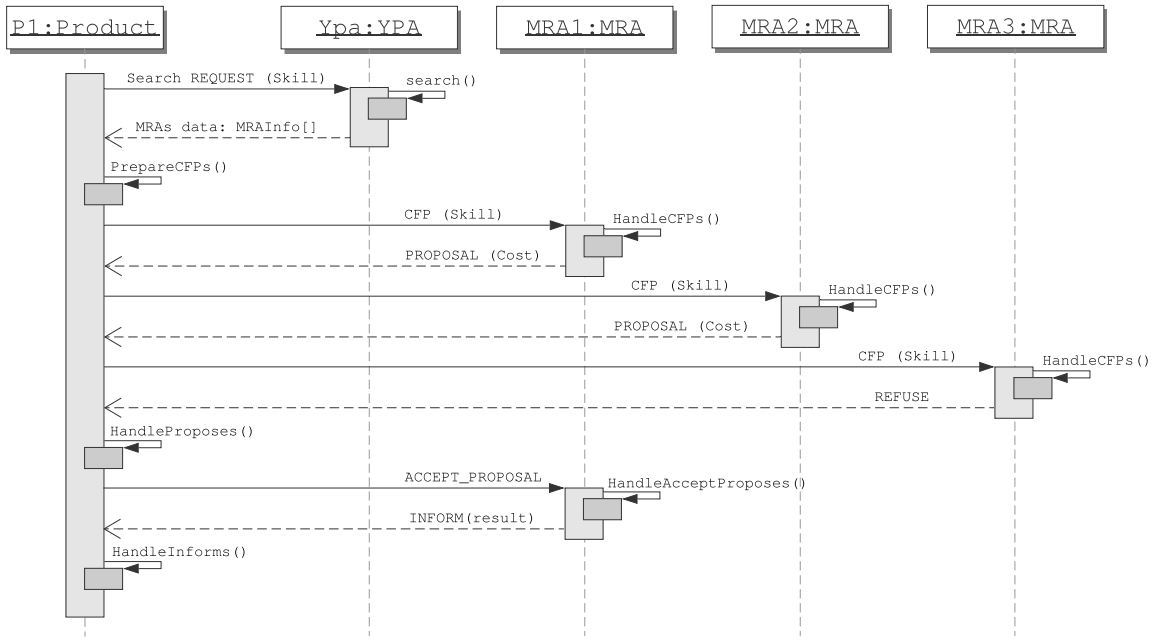
Fonte: De autoria própria.

4.1.2 Rede de contratos e execução remota

A fim de evitar ter de implementar um método para responder outros agentes em cada agente cyber-físico, decidiu-se estabelecer um método padrão na classe base desses agentes (isto é, na classe `MRA`) no qual adiciona-se um comportamento respondedor. Assim, aproveitou-se para estabelecer um alinhamento com o protocolo FIPA Contract Net Interaction, uma vez que sistemas manufatura podem ter mais de um módulo mecatrônico que desempenham a mesma função, podendo refletir em mais de um agente cyber-físico disponibilizando a mesma skill. Tal método é o `addResponderBehaviour()` e, na prática, ele implementa o papel de um "participante"no protocolo citado.

Para completar a rede de contratos, o método `newRemoteExecuteBehaviour()`, que recebe somente uma `skill`, foi criado para implementar o papel de um "iniciador"no protocolo citado. Logo, tal método permite ao agente iniciar uma rede contrato para executar uma dada skill. Além disso, o método encapsula a busca pelos agentes através da solicitação de busca feita para o `YPA`. A figura 11 ilustra a comunicação desencadeada ao ser feita uma execução remota de uma determinada skill.

Figura 11 – Comunicação em uma execução remota.



Fonte: De autoria própria.

É importante citar que tais métodos dependem de outros métodos da classe `MRA`, os quais especificam uma série de maneiras de como serão tratadas as mensagens que serão recebidas em meio à essa rede. Tais maneiras poderiam ser especificadas nos métodos principais, mas ao serem especificadas em métodos protegidos separados, o usuário, ao criar uma classe filha para sua aplicação, pode sobrescrever tais métodos de forma a mudar completamente tais maneiras de acordo com suas necessidades.

4.1.3 Cliente Modbus TCP/IP

Como mostrado na Figura 10, foi inserido um objeto cliente Modbus TCP/IP na classe `MRA`, assim como os métodos que encapsulam as interações com esse objeto, de forma a facilitar o uso. Existe um método para o setup padrão do cliente, onde é feita a conexão com o server, e os métodos de escrita/leitura de variáveis do servidor.

Os métodos `writeCoil()` e `readInput()`, respectivamente, são responsáveis por encapsular a escrita e leitura de variáveis do tipo booleanas (ou seja, bits), normalmente associadas aos triggers de solicitação de serviços ou sinais de confirmação de término da execução dos mesmos. Já os métodos `writeHoldingRegister()` e `readHoldingRegister()` encapsulam a escrita e leitura de variáveis do tipo `int`, associadas à posição e identificação de cor.

4.1.4 Classe de mapeio Modbus

Implementou-se também uma classe responsável por guardar os endereços a serem usados nas leituras e escritas no servidor Modbus. Dessa forma, obteve-se uma forma mais organizada e legível de lidar com o mapeamento de endereços e labels do CLP, dentro do código Java. Essa classe e suas variáveis do tipo `static` são usadas na hora de instanciar os agentes, os quais precisam de valores de endereço a serem usados nos métodos de seus respectivos clientes Modbus durante a execução de suas skills.

O trecho de código abaixo mostra um exemplo de mapeamento para as variáveis de uma esteira, a qual precisa dos endereços das variáveis de gatilho de execução dos serviços de mover, receber e enviar; assim como os endereços das variáveis de confirmação de cada serviço. Os gatilhos de execução são *coils*, enquanto as confirmações são *inputs*, por isso, é possível usar os mesmos valores do tipo `int` para 'mover' e 'moveu', por exemplo.

```
1  public class ModbusMapping {
2      //Conveyor 1
3      public static final int CONVEYOR_1_MOVE = 0;
4      public static final int CONVEYOR_1_RECEIVE = 1;
5      public static final int CONVEYOR_1_SEND = 2;
6      public static final int CONVEYOR_1_MOVED = 0;
7      public static final int CONVEYOR_1_RECEIVED = 1;
8      public static final int CONVEYOR_1_SENT = 2;
9
10     {...}
11 }
```

Código 4.1 – Exemplo de classe de mapeio Modbus. Fonte: De autoria própria.

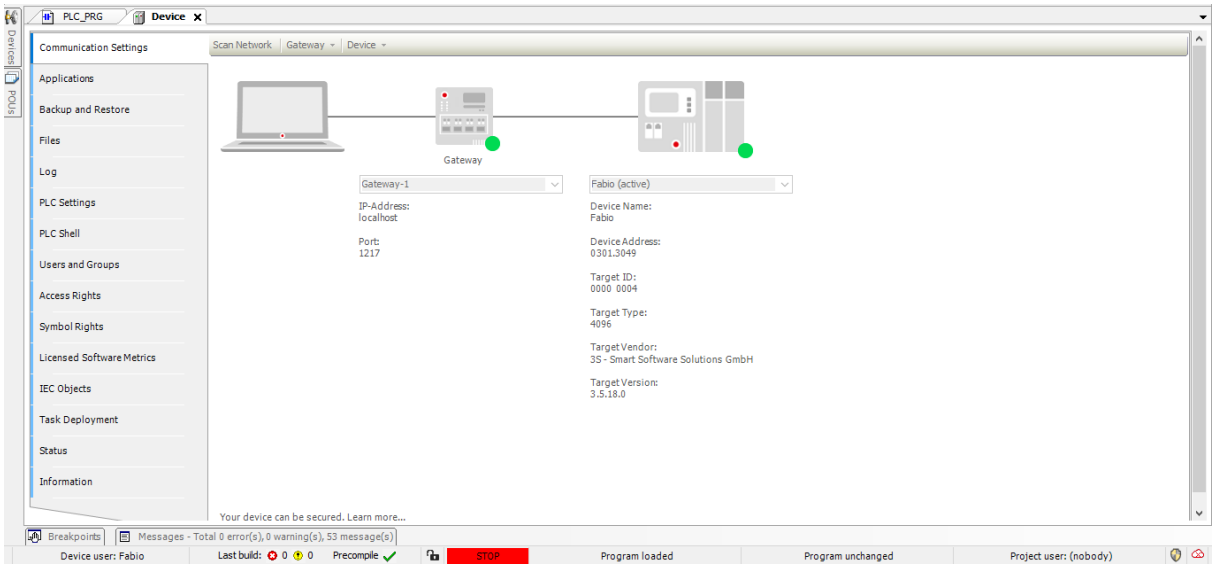
4.2 CLP virtual

Como dito anteriormente, para a arquitetura de integração proposta, foi utilizado o ambiente de desenvolvimento CODESYS, que permite o desenvolvimento de sistemas de automação nas linguagens padrão IEC-61131-3. Assim, pode ser descrito em Ladder e FBD o comportamento dos módulos de uma planta, visando, uma programação orientada a serviços, os quais representam as skills na arquitetura EPSCore. Utilizando o CODESYS, é possível, ainda, tomar vantagem da programação orientada a objeto, implementando uma `POU` para cada módulo, que pode ter várias instâncias no sistema (uma esteira, por exemplo).

Ao fazer a instalação padrão da IDE (V3.5 SP18 - Windows 10), é instalado um CLP virtual que é configurável pela IDE, podendo ser iniciado e parado (comando *Run e Stop*) pelo

ícone no menu de ícones oculto da barra de tarefas do Windows 10. A Figura 12 mostra o CLP conectado no CODESYS.

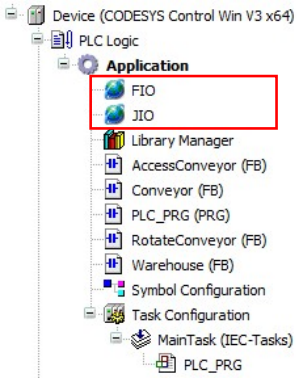
Figura 12 – CLP virtual configurável conectado no CODESYS.



Fonte: De autoria própria.

Além disso, existe a fácil implementação dos servidores que se comunicam com a camada superior (camada física da arquitetura EPSCore) e com a inferior (planta virtual). Ao configurar tais servidores, é possível criar uma lista exclusivamente para o servidor utilizar as variáveis (labels) dessa no código Ladder/FBD, de maneira que a IDE implemente um amarro dessas variáveis aos servidores e o programador possa usá-las de forma organizada. Foram criadas duas listas de variáveis globais (*Global List Variable* na IDE: FIO (Factory Input Output) e JIO (Java Input Output).

Figura 13 – Lista de variáveis globais criadas para a arquitetura.

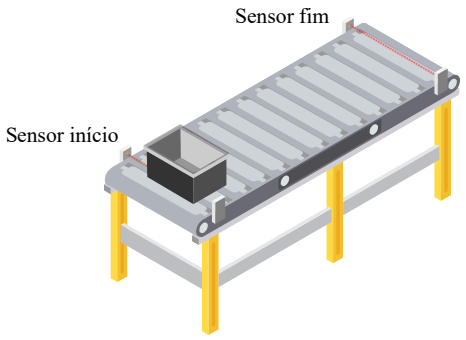


Fonte: De autoria própria.

4.2.1 Programação Ladder/FBD

No CODESYS é especificado o comportamento dos módulos através das linguagens de programação de acordo com a IEC-61131-3. Assim, pode-se criar as `POU`s (equivalente à classe) que modelam o comportamento de cada módulo, e instanciar objetos das mesmas de acordo com a necessidade da planta. Um detalhe importante é a forma como esse comportamento é modelado, uma vez que a intenção final é a chamada dos serviços da planta pelos agentes cyber-físicos. Logo, tal especificação é feita de modo a ofertar as ações das skills dos agentes de cada módulo.

Figura 14 – Esteira exemplo com dois sensores.



Fonte: De autoria própria.

Como mostrado na Figura 14, no caso de uma esteira simples, com dois sensores (início e fim), e um serviço de mover um caixote do início até o fim, poderia-se criar uma `POU` denominada `Conveyor`, a qual teria como entrada:

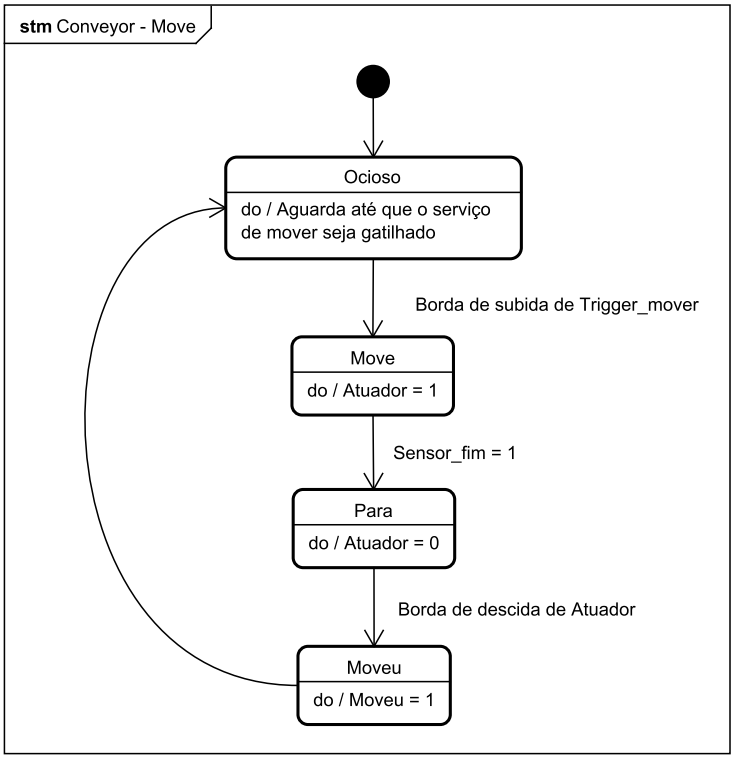
- `EN`: variável do tipo `bool` que representa o *enable* do módulo;
- `Sensor_inicio`: variável do tipo `bool` que representa o bit associado ao sensor do início da esteira;
- `Sensor_fim`: variável do tipo `bool` que representa o bit associado ao sensor do fim da esteira;
- `Trigger_mover`: variável do tipo `bool` que representa o bit associado ao gatilho de solicitação do serviço de mover da esteira;

E saídas:

- `Atuador`: variável do tipo `bool` que representa o atuador da esteira;
- `Moveu`: variável do tipo `bool` que informa que o serviço de mover foi finalizado.

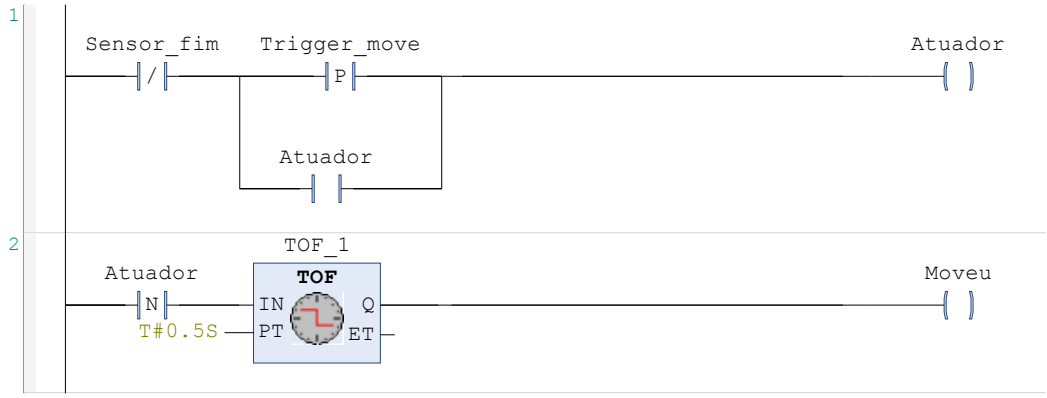
Implementando, portanto, o seguinte comportamento ilustrado da máquina de estado da Figura 15, do qual se observa o módulo em questão ofertando o serviço de mover um caixote que está na posição de início (marcada pelo sensor de início) até a posição final (marcada pelo sensor de fim), tem-se o diagrama Ladder da `POU` na Figura 16.

Figura 15 – Máquina de estado da esteira com o serviço de mover.



Fonte: De autoria própria.

Figura 16 – Diagrama Ladder da POU de uma esteira com o serviço de mover.

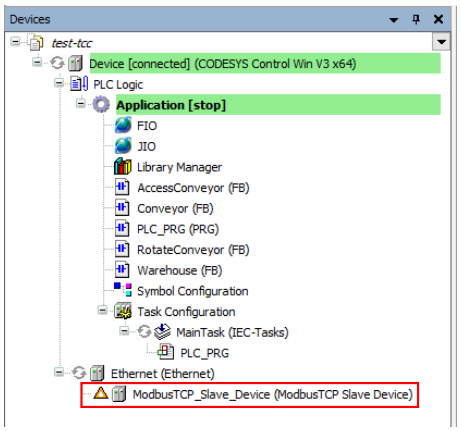


Fonte: De autoria própria.

4.2.2 Servidor Modbus TCP/IP

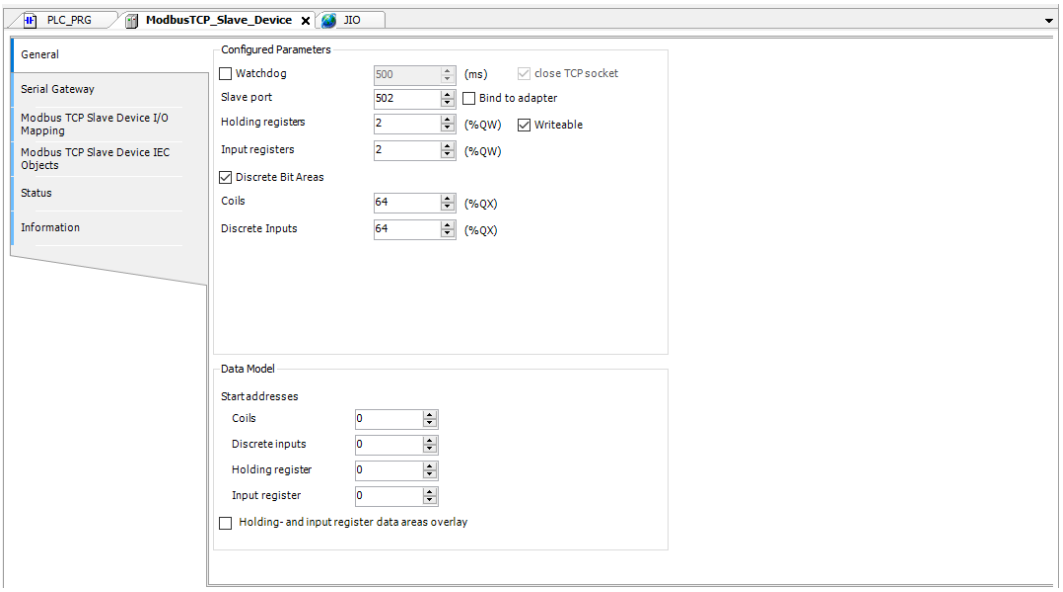
Para implementar um servidor Modbus no CLP, é adicionado à lista de dispositivos um servidor Modbus, conforme mostrado na Figura 17, a partir do qual pode-se configurar a porta, número de bobinas, registradores, entradas e saídas (Figura 18), além do mapeamento para variáveis globais (Figura 19), as quais estarão amarradas às do servidor.

Figura 17 – Servidor Modbus TCP/IP.



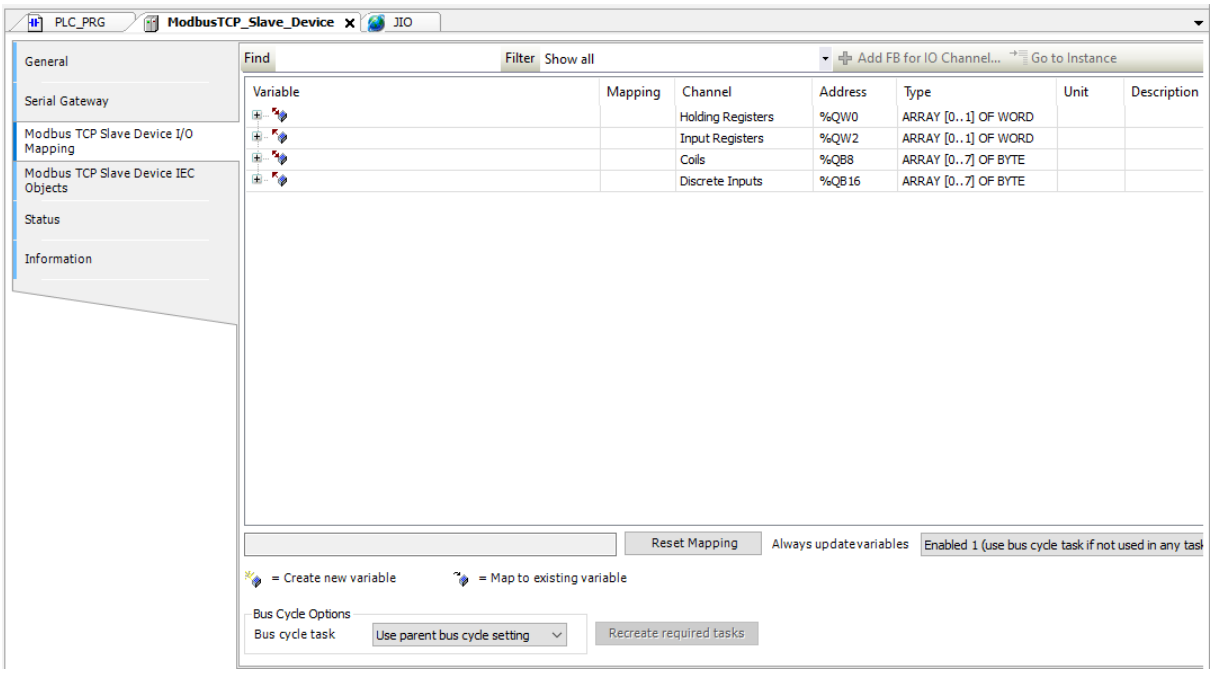
Fonte: De autoria própria.

Figura 18 – Tela de configuração geral do dispositivo Modbus TCP/IP.



Fonte: De autoria própria.

Figura 19 – Tela de mapeamento de variáveis do dispositivo Modbus TCP/IP.

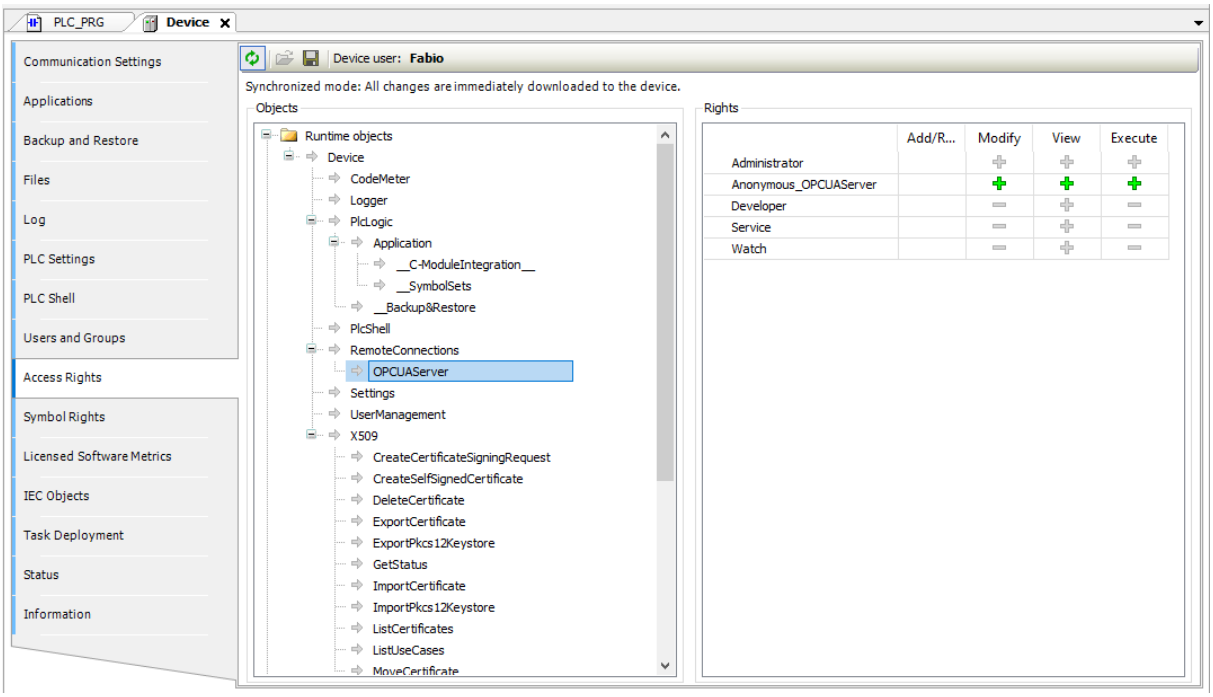


Fonte: De autoria própria.

4.2.3 Servidor OPC UA

Por padrão, a versão 3.5 SP18 do CODESYS implementa um servidor OPC-UA no CLP virtual, não necessitando, portanto, adicionar um dispositivo, nem ativar. Assim, a única configuração feita foi na tela de permissão de acesso (Figura 20) das variáveis, às quais foram garantidos todos os acessos, para que o cliente no Factory I/O possa interagir livremente.

Figura 20 – Tela de configuração de acesso do servidor OPC-UA.

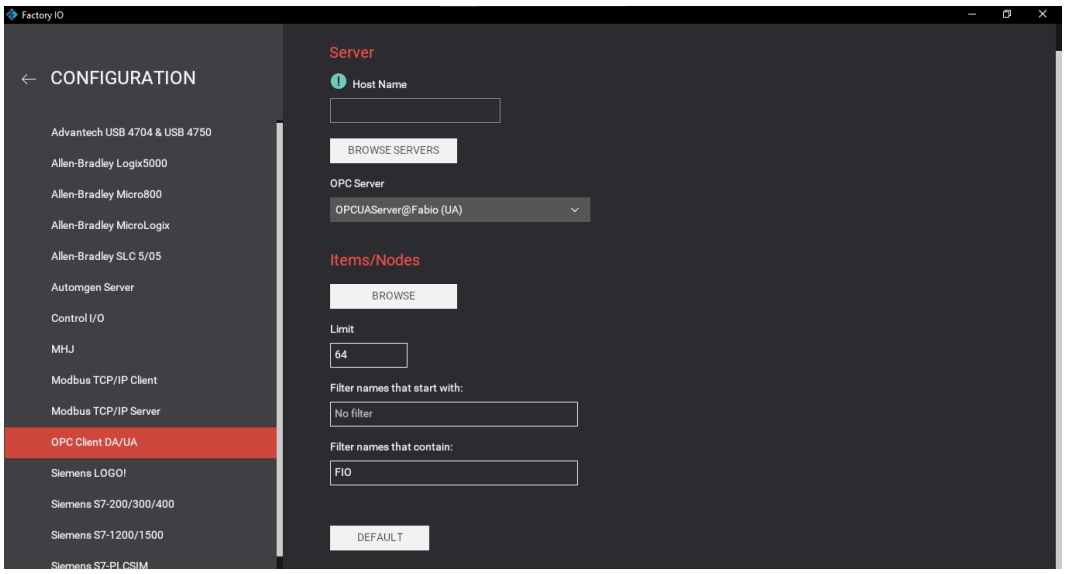


Fonte: De autoria própria.

4.3 Planta virtual

Dentro do software Factory I/O (versão 2.4.3) é implementado uma planta industrial a ser controlada pelo sistema multiagente. Cada módulo da planta, com suas respectivas entradas e saídas, são amarradas aos nós do cliente OPC-UA configurado na seção de *Drivers*, como mostra a Figura 21. A Figura 22 mostra um exemplo de conexão das entradas e saídas com o cliente.

Figura 21 – Tela de configuração do cliente OPC-UA no Factory I/O.



Fonte: De autoria própria.

Figura 22 – Exemplo de tela de conexão das entradas e saídas ao cliente OPC-UA.

Fonte: (Factory IO, 2023)

5 Estudo de caso

Esta seção mostra um exemplo de aplicação e sua implementação na arquitetura proposta para a visualização e simulação de um processo de manufatura utilizando um sistema multiagente com a arquitetura EPSCore. Para tal, tomou-se como base uma planta real, que é um demonstrador didático de automação industrial. Além disso, elaborou-se um comportamento a ser alcançado para o processo industrial e, para alcançá-lo, foi necessário implementar algumas classes que estendem as funcionalidades padrão da arquitetura EPSCore. A seção em questão objetiva, também, esclarecer o desenvolvimento de tal descrição.

5.1 Modificações na arquitetura

5.1.1 Plano de execução

Os agentes do tipo `Product` desempenham seus papéis no sistema através da execução de seus respectivos processos (método `produce()`), os quais nada mais são que um conjunto de chamadas remotas de skills ou de outros processos (isto é, outros agentes do tipo `Product`).

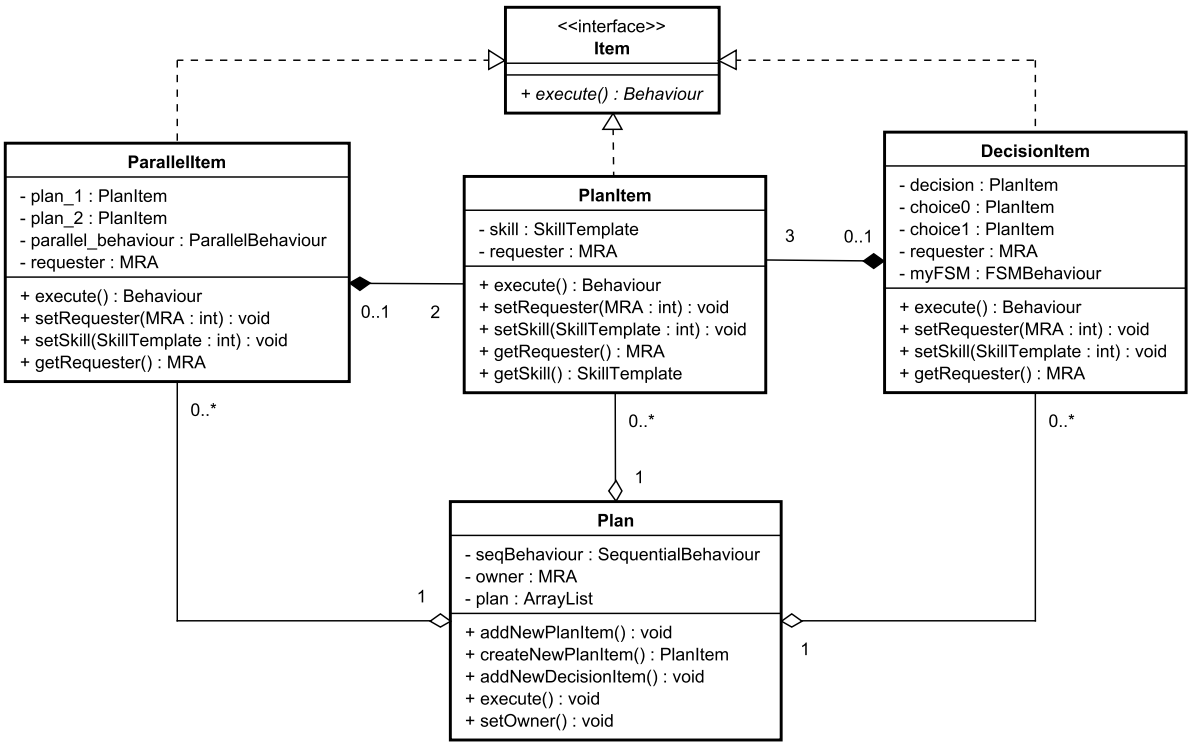
Na descrição, esses agentes fazem chamadas remotas de skills de agentes da camada física. E, a fim, de organizar tal conjunto de chamadas, foram criadas classes que implementam um plano de execução na arquitetura. São elas:

- `Item`: é uma interface que tem somente um método: `execute()`. Tal método deve ser implementado de forma a retornar um comportamento que realiza a execução de um item num plano de execução;
- `PlanItem`: classe que implementa a interface `Item` e define um item simples dentro de um plano de execução. Tem um atributo do tipo `skill`, o qual define a skill a ser executada, e um atributo do tipo `MRA`, o qual define o agente pelo qual o item será executado (isto é, o dono do plano no qual o item está inserido). O método `execute()` é definido como a execução remota da skill citada por parte do agente também citado;
- `DecisionItem`: classe que implementa a interface `Item` e define um item de decisão dentro de um plano de execução através de 3 objetos do tipo `PlanItem`: um para decisão e dois para escolhas. No método `execute()` cria e retorna-se um objeto do tipo `FSMBehaviour`, no qual a execução da decisão é o estado inicial e as execuções das escolhas são os estados finais. Caso o resultado da decisão seja 0, a escolha 0 (`choice0`) se segue. Caso seja 1, a escolha 1 (`choice1`) é escolhida;

- `ParallelItem`: classe que implementa a interface `Item` e define um item de execução paralela de duas skills dentro de um plano de execução através de 2 objetos do tipo `PlanItem`. No método `execute()` cria e retorna-se um objeto do tipo `ParallelBehaviour`, o qual executa as duas skills paralelamente;

- `Plan`: define um plano de execução no sistema e provém métodos que encapsulam todo o processo de execução para serem usados finalmente na classe `Product`. Tal classe tem um atributo do tipo `Plan` e esse plano é criado utilizando os métodos de adicionar itens (`addNewPlanItem()` e `addNewDecisionItem()`). Ao serem criados, tais itens são guardados num `ArrayList` e, uma vez que o plano seja executado, cada item tem seu comportamento de execução extraído e guardado num `SequentialBehaviour`, o qual é adicionado ao agente dono do plano (`Owner`).

Figura 23 – Classes que implementam um plano de execução na descrição.

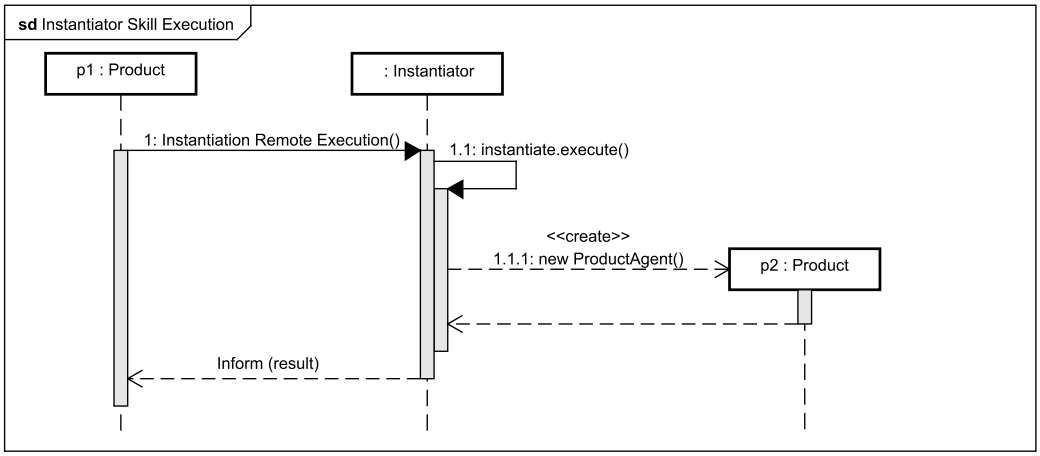


Fonte: De autoria própria.

5.1.2 Agente Instanciador

A fim de garantir uma sociedade dinâmica de agentes no sistema, foi criado um agente capaz de ofertar o serviço de instanciar (criar) agentes. Dessa forma, essa skill é oferecida aos agentes cognitivos, os quais podem escolher instanciar agentes conforme a necessidade do sistema. O processo de oferta de tal skill é mostrado no diagrama de sequência da Figura

Figura 24 – Comunicação do serviço de instanciar.

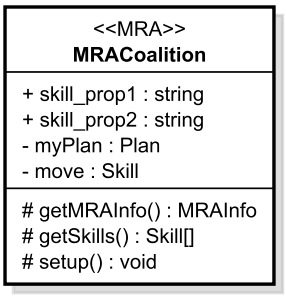


Fonte: De autoria própria.

5.1.3 Agente de Coalisão de MRAs

Um agente de coalisão se mostrou necessário ao descrever as skills de mover entre os módulos, uma vez que, para tal, é necessário sincronia e paralelismo entre as execuções de skill dos dois agentes em questão, os quais representam os módulos. Por exemplo, ao passar um caixote de uma esteira para outra, é necessário que a primeira esteira envie o caixote (i.e. motor seja acionado) e que a segunda esteira receba tal caixote (i.e. motor seja acionado) simultaneamente.

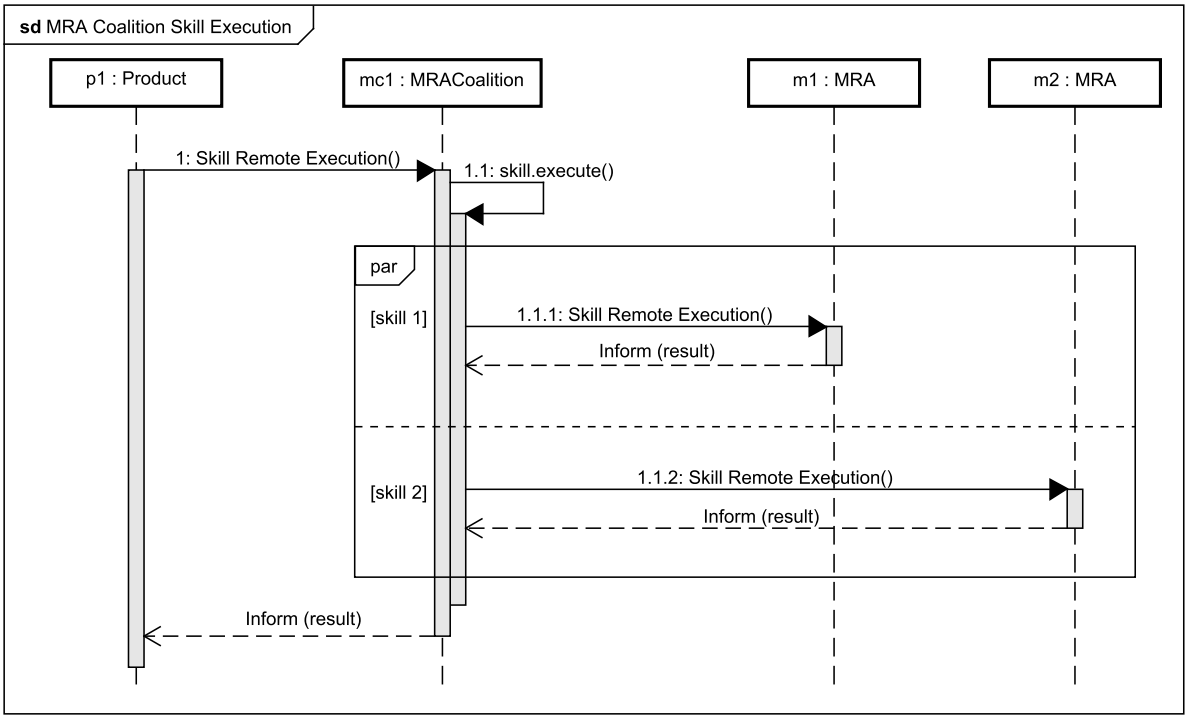
Figura 25 – Classe que descreve um agente de coalisão de MRAs



Fonte: De autoria própria.

A figura 25 mostra os métodos e parâmetros da classe do agente de coalisão de MRA, que é um MRA que tem um plano de execução responsável por executar remotamente duas skills em paralelo. A figura 26, por sua vez, demonstra a sequência da comunicação de um agente de coalisão de MRAs ofertando a skill de execução paralela de duas skills de dois MRAs. É papel, portanto, desse agente, encapsular a execução de skill de dois agentes MRAs, assim, um agente do tipo produto pode solicitar o serviço de dois MRAs ao mesmo tempo.

Figura 26 – Sequência da execução de skill de um agente de coalisão de MRAs

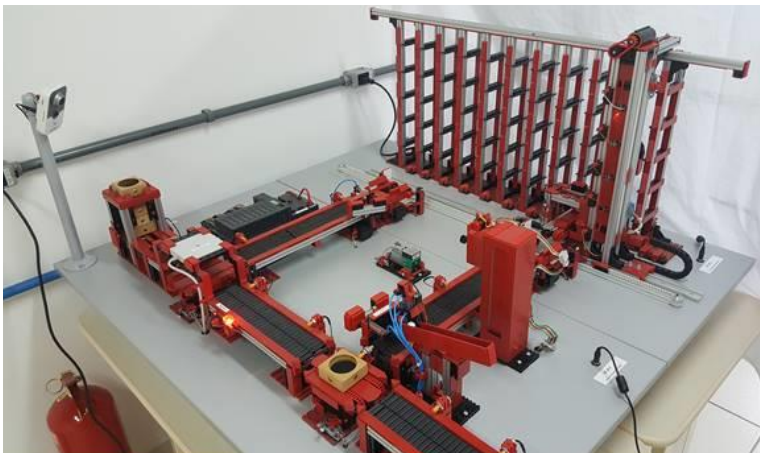


Fonte: De autoria própria.

5.2 Planta Staudinger

Foi estudado um cenário aplicável de um EPS/EAS, para o qual o projeto se propõe a aplicar a arquitetura EPSCore, tais quais as análises. O objeto de estudo foi o protótipo de um sistema de manufatura que tem como objetivo principal simular uma linha de produção industrial automática, utilizando esteiras, sensores e atuadores, fazendo uso da intervenção humana somente em seu controle e programação por meio de controlador lógico programável (CLP).

Figura 27 – Conjuto Staudinger.

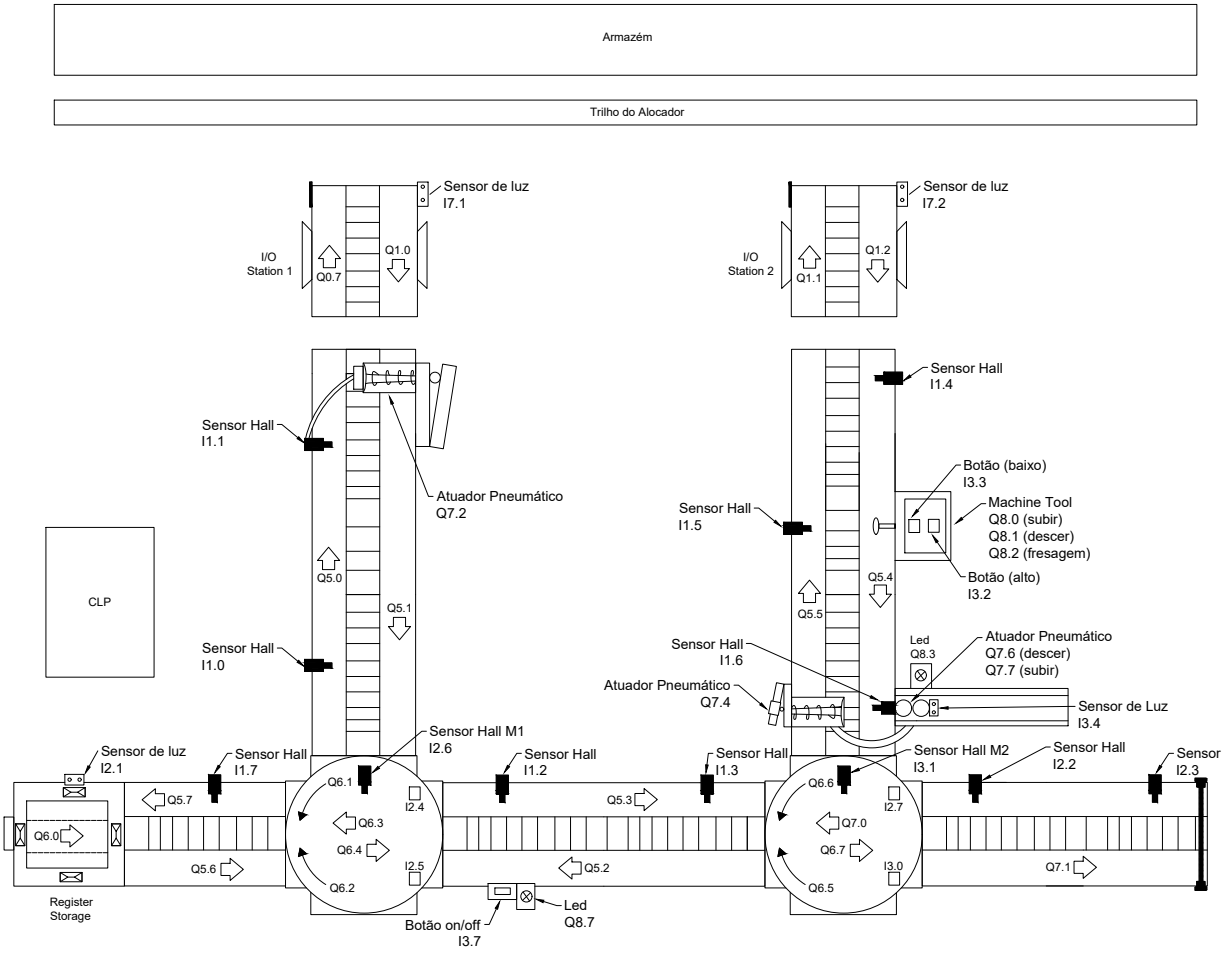


Fonte: De autoria própria.

O protótipo se chama conjunto Staudinger e é formado por módulos que cumprem funções específicas. São eles:

- Resgister Storage with Conveyor Belt: é responsável por ser o começo da produção. Para tal, possui uma pilha de caixotes, da qual os mesmos são retirados para serem inseridos na linha de produção;
- Rotate Conveyor Belt: é composto por esteiras rotativas. Possui a função de fornecer direções diferentes aos caixotes. Para tal, avalia a cor do caixote (por meio de seus sensores) e o redireciona à esteira (próximo módulo) adequada.
- High Level Storage Warehouse: Este modelo é composto por uma esteira transportadora, outras duas de acesso, elevador e armazém para as peças. A função deste modulo é simular um processo de armazenagem de produto, algo bem comum na indústria;
- Conveyor Belt with Machine Tool: é composto por uma esteira transportadora e um atuador capaz de subir e abaixar uma ferramenta. A função deste conjunto é simular algum tipo de mecanismo existente na indústria, como uma prensa ou um cortador;
- Pneumatic picking: é composto por uma esteira transportadora e um atuador pneumático capaz de inserir pequenas esferas nos caixotes. A função deste modulo é simular a inserção de algum recurso existente na indústria, como, por exemplo, bombons em uma caixa de chocolates.

Figura 28 – Planta do Staudinger.



Fonte: De autoria própria.

Por ser controlado por meio de um controlador lógico programável (CLP), o Staudinger admite uma série de possibilidades de comportamentos, desde que o programador os implementem. Tal fato permite também que o Staudinger possa disponibilizar microsserviços, como, por exemplo, o de mover uma determinada esteira por um determinado tempo. Com isso, o funcionamento geral pode ser fragmentado em pequenos serviços que poderão ser executados sempre que um outro sistema (um sistema multiagente, no caso deste trabalho) os solicitem. É a partir disso que a arquitetura EPSCore poderá ser aplicada.

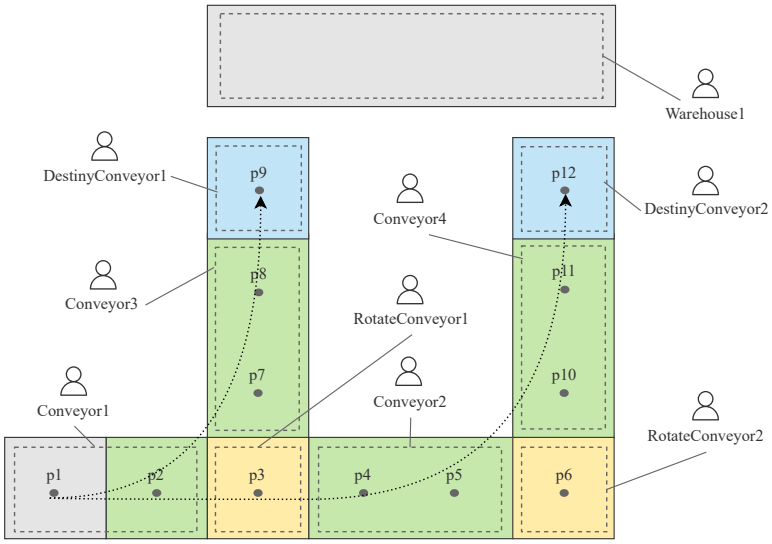
5.3 Comportamento adotado

Para este trabalho, foi utilizado uma planta virtual inspirada no demonstrador Staudinger e o comportamento adotado é um sistema simplificado que consiste basicamente de uma produção com segregação, onde o produto é um caixote com uma peça dentro que pode ser da cor azul ou verde. O sistema permite a solicitação de um produto (um caixote com uma peça de uma cor específica), a partir da qual um caixote com peça de cor aleatória é criado. Um módulo

de reconhecimento de cor identifica se o caixote tem a peça da cor especificada na solicitação e, caso a cor seja coincida, o caixote segue um caminho (nomeado de produção); caso não, o produto segue outro caminho (nomeado de inserção).

Assim, os destinos dos caixotes são as esteiras de acesso ao armazém (esteiras de destino), que são duas (uma para cada caminho). Ao chegar a alguma dessas esteiras, um sistema de armazém vertical automático se encarrega de guardar o caixote. O usuário final poderá requisitar uma série de produtos de uma vez. Ao solicitar uma produção, o usuário especificará a cor desejada.

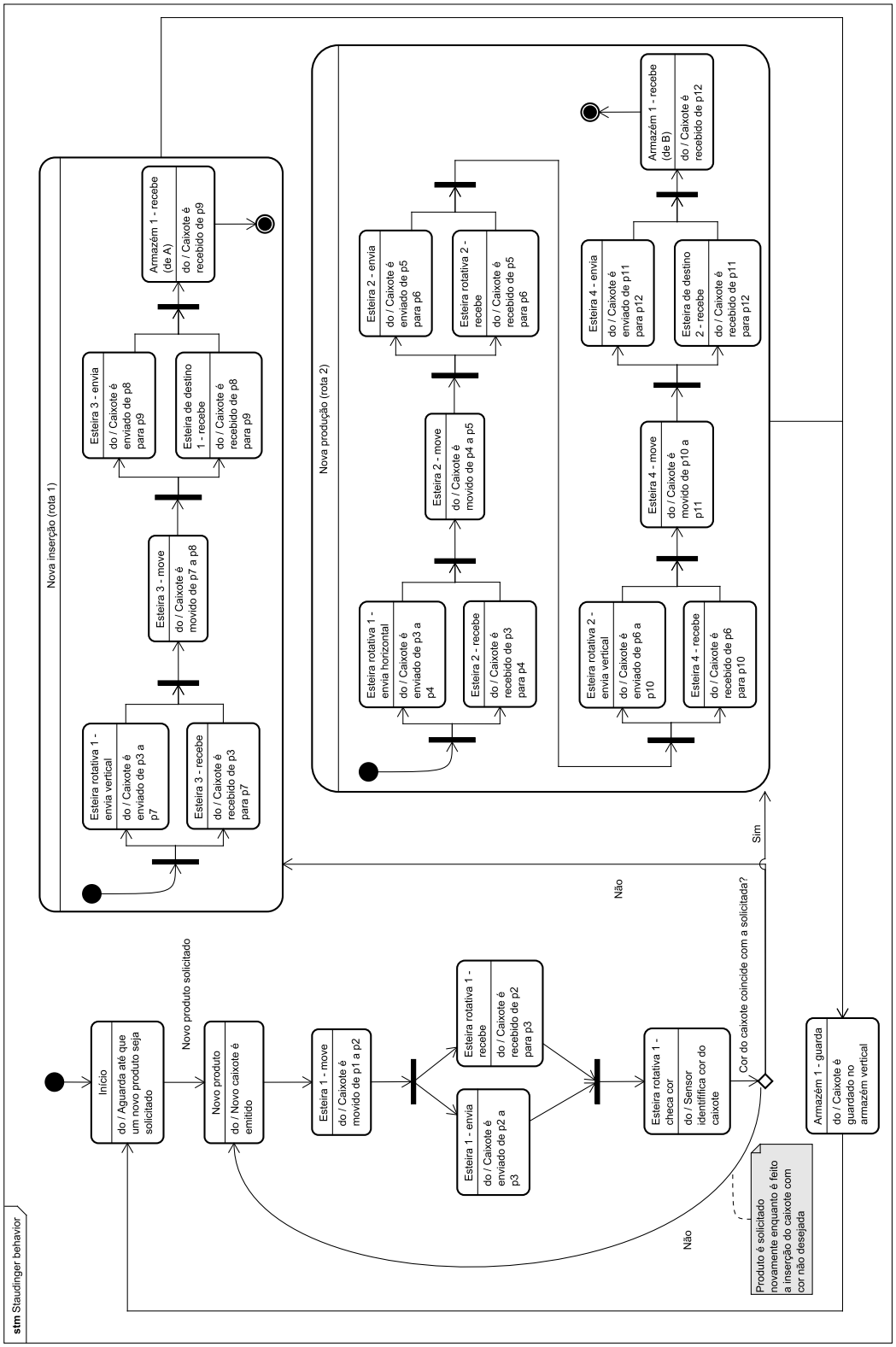
Figura 29 – Rotas e mapeamento do sistema no comportamento adotado.



Fonte: De autoria própria.

Como pode ser visto na Figura 29, foi feito um mapeamento das posições no sistema, o qual é modular e tem um agente para cada módulo. É possível ver também as rotas dos caixotes: inserção (rota mais à esquerda - A) e produção (mais à direita - B).

Figura 30 – Fluxo geral do comportamento do sistema

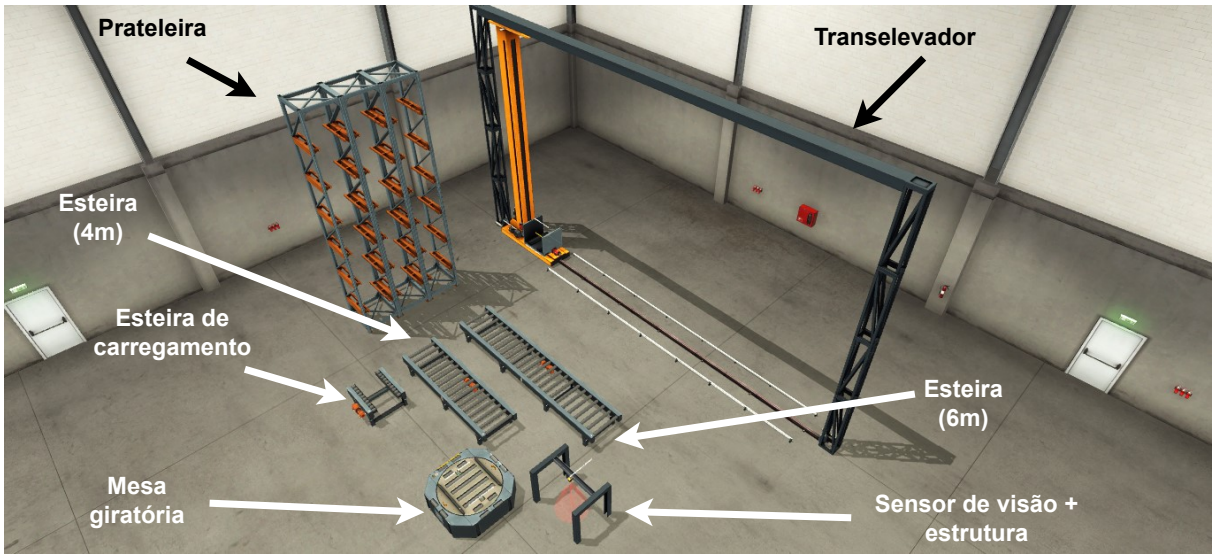


Fonte: De autoria própria.

5.4 Planta virtual

Para a planta virtual, foram utilizados os módulos nativos do Factory I/O: esteira de 4m, esteira de 6m, mesa giratória, esteira de carregamento, transelevador e prateleira. Além disso, foram utilizados sensores difusos nas esteiras e um sensor de visão na primeira mesa giratória. Os módulos podem ser vistos na Figura

Figura 31 – Módulos utilizados no Factory I/O.

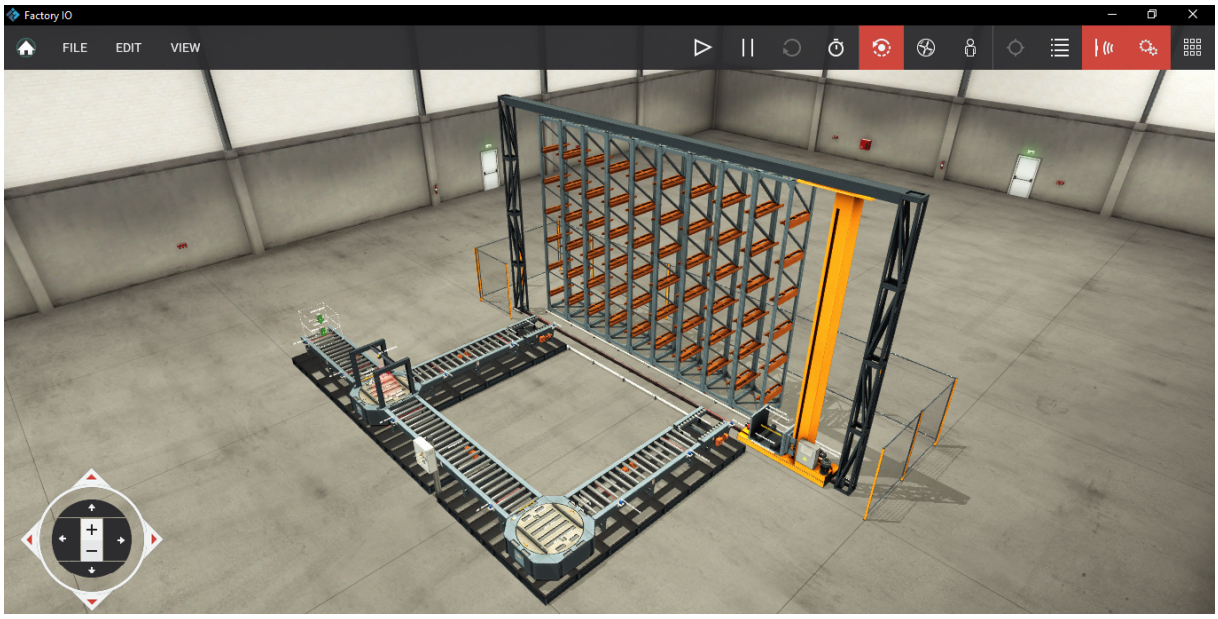


Fonte: De autoria própria.

Ao montar a planta, foi adicionado dois objetos emissores, um para gerar caixotes empilháveis, e outro para gerar o conteúdo do produto, que é chamado de "parte"no Factory I/O e trata-se de um material que pode ser da cor azul, cinza ou verde. A escolha da cor a ser emitida foi definida como aleatória entre verde e azul.

Por fim, a mesa giratória servirá para modelar a esteira rotativa; a esteira de carregamento modelará a esteira de destino; o transelevador em conjunto com algumas prateleiras modelará o armazém, e os caixotes são emitidos de acordo com a produção. A planta final montada pode ser vista na Figura 32.

Figura 32 – Planta final montada no Factory I/O.



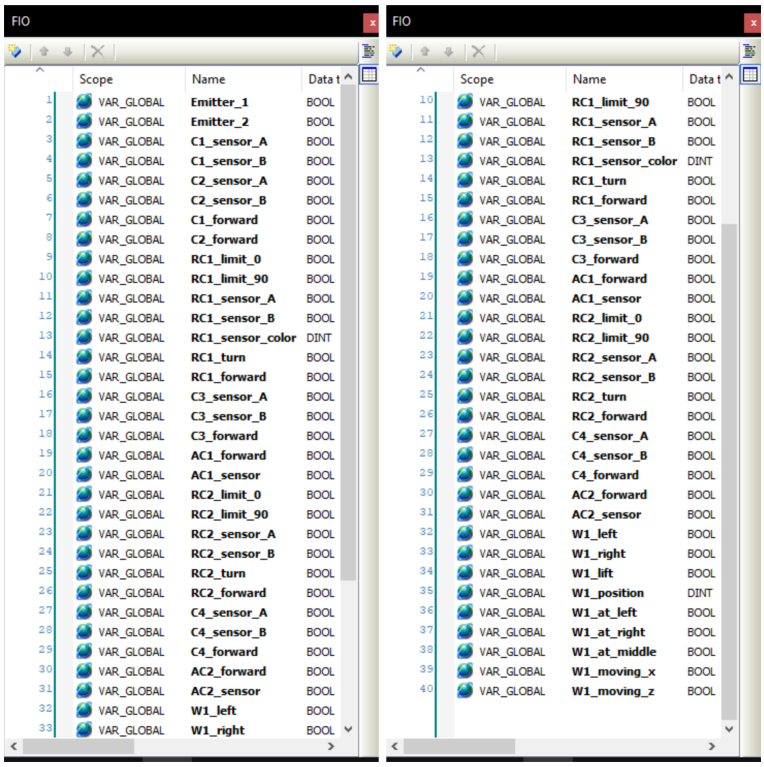
Fonte: De autoria própria.

5.5 Controle do processo usando o CLP

No CODESYS, foram desenvolvidas algumas POUs para o sistema. Especificamente, uma para cada tipo de módulo, contemplando a implementação dos serviços na planta física que são encapsulados e ofertados pelos agentes do sistema. Os serviços são ofertados com uma variável booleana de entrada, cujo pulso positivo inicia o processo. Na saída, são expostas variáveis de confirmação do fim da execução do serviço, uma vez que o agente cyber-físico precisa saber quando o serviço terminou.

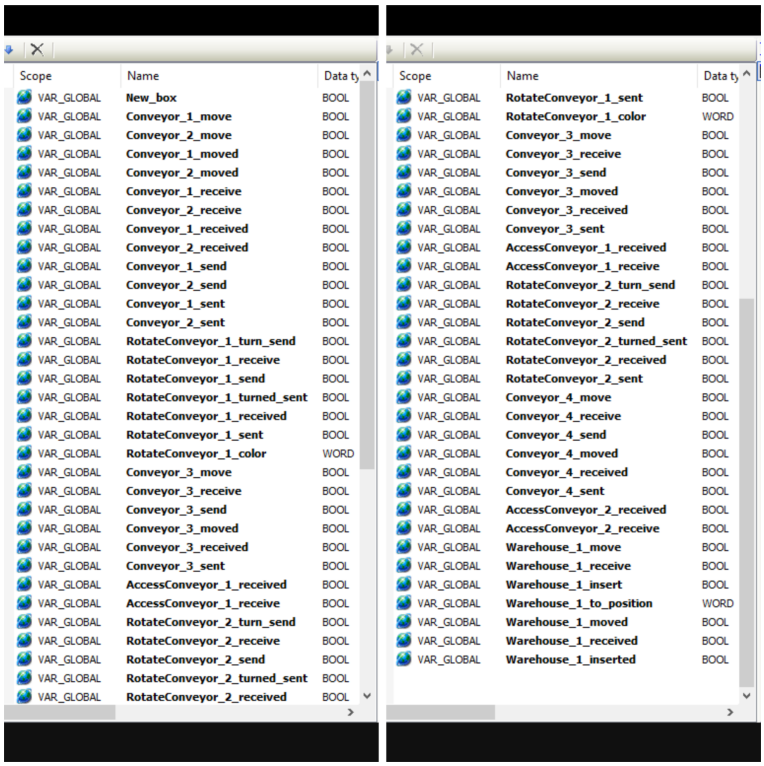
Foi feita também a configuração do servidor Modbus TCP/IP, da lista de variáveis globais para lidar com o sistema Java e da lista para lidar com o Factory I/O. A configuração é mostrada na Figura, enquanto a lista de variáveis de entradas e saídas que lida com o Factory I/O é mostrada na Figura e a que lida com o sistema Java na Figura

Figura 33 – Lista de variáveis globais de entradas/saídas com o Factory I/O.



Fonte: De autoria própria.

Figura 34 – Lista de variáveis globais de entradas/saídas com a aplicação Java.



Fonte: De autoria própria.

5.5.1 Conveyor

A POU `Conveyor` modela os módulos de esteira e implementa o serviço de receber o caixote para a posição inicial, mover da posição inicial para a final e de enviar o caixote da posição final para o próximo módulo. Para tal, ela possui as seguintes variáveis de entrada:

- `Trigger_move`: responsável por receber o sinal de gatilho para a execução do serviço de mover, o qual move um caixote da posição inicial (sensor A) para a final (sensor B) (referência da Figura 14);
- `Stop_move_forward`: responsável por receber o sinal do sensor de fim da esteira, de forma a saber que o caixote chegou no destino;
- `Trigger_receive`: responsável por receber o sinal de gatilho para a execução do serviço de receber, o qual move a esteira de forma a receber (para posição inicial) um caixote vindo de um módulo anterior;
- `Stop_receive_forward`: responsável por receber o sinal do sensor de início da esteira, de forma a saber que o caixote chegou no módulo;
- `Trigger_send`: responsável por receber o sinal de gatilho para a execução do serviço de enviar, o qual move a esteira de forma a enviar (da posição final) um caixote para um módulo posterior;
- `Stop_send_forward`: responsável por receber o sinal do sensor de início da esteira posterior, de forma a saber que o caixote chegou no módulo posterior (foi enviado).

Para as variáveis de saída, tem-se:

- `Running_forward`: saída que controla o atuador da esteira, de modo a ligar ou desligar a mesma;
- `Moved`: saída de confirmação da finalização do serviço de mover;
- `Received`: saída de confirmação da finalização do serviço de receber;
- `Sent`: saída de confirmação da finalização do serviço de enviar;

5.5.2 Rotate Conveyor

A POU `Rotate_Conveyor` modela os módulos de esteira rotativa e implementa o serviço de receber o caixote, enviar para o módulo posterior horizontalmente e de enviar para o módulo posterior verticalmente. Como no sistema utilizado como estudo de caso a esteira

rotativa só possui duas possibilidades de envio do caixote, preferiu-se dividir em horizontal e vertical, sendo o envio horizontal o processo que requer que a esteira rotacione 90º.

A POU possui as seguintes variáveis de entrada:

- `Trigger_turn_send`: responsável por receber o sinal de gatilho para a execução do serviço de girar e enviar (envio vertical), o qual move um caixote para o módulo posterior que está acessível após o giro de 90º da esteira rotativa;
- `Stop_turn_send`: responsável por receber o sinal do sensor de início do módulo posterior localizado à 90º, de forma a saber que o caixote chegou no destino;
- `Trigger_receive`: responsável por receber o sinal de gatilho para a execução do serviço de receber, o qual move a esteira de forma a receber um caixote vindo do módulo anterior (na aplicação, só se recebe vindo de uma posição);
- `Stop_receive`: responsável por receber o sinal do sensor da esteira rotativa, de forma a saber que o caixote chegou na mesma;
- `Trigger_send`: responsável por receber o sinal de gatilho para a execução do serviço de somente enviar (envio horizontal), o qual move um caixote para o módulo posterior (que está acessível sem o giro) à esteira rotativa;
- `Stop_send_forward`: responsável por receber o sinal do sensor de início do módulo posterior (que está acessível sem o giro), de forma a saber que o caixote chegou no destino;
- `Is_vertical`: responsável por receber o sinal do sensor de curso 90º da esteira rotativa, de forma a informar se a esteira rotativa está na posição vertical;
- `Is_horizontal`: responsável por receber o sinal do sensor de curso 0º da esteira rotativa, de forma a informar se a esteira rotativa está na posição horizontal;

Para as variáveis de saída, tem-se:

- `Running_forward`: saída que controla o atuador da esteira presente na esteira rotativa, de modo a ligar ou desligar a mesma;
- `Turn`: saída que controla o atuador de giro da esteira rotativa, de modo a girar 90º (ou não) a mesma;
- `Turned_sent`: saída de confirmação da finalização do serviço de envio vertical;
- `Sent`: saída de confirmação da finalização do serviço de envio horizontal;
- `Received`: saída de confirmação da finalização do serviço de receber;

5.5.3 Access Conveyor

A POU `Access_Conveyor` modela o comportamento da esteira de acesso ao armazém, que também foi denominada de esteira de destino nas seções anteriores. Essa esteira é mais simples, pois só recebe o caixote, uma vez que o transelevador pega o caixote dela. Portanto, essa POU tem somente duas entradas: o gatilho para o serviço de receber e a entrada de parada do recebimento, isto é, para receber o sinal do sensor da mesma. As saídas também são somente duas: uma para o atuador da esteira e outra para a confirmação de finalização do serviço.

5.5.4 Warehouse

A POU `Warehouse` modela o comportamento do armazém, o qual tem 3 serviços no sistema: mover o elevador para uma determinada posição, pegar o caixote e inseri-lo. Portanto, o módulo armazém representa o conjunto de prateleiras junto com o transelevador.

As entradas da POU são:

- `Trigger_move`: responsável por receber o sinal de gatilho para a execução do serviço de mover, o qual move o transelevador para a posição especificada em `To_position`;
- `To_position`: responsável por especificar a posição de destino do serviço de mover (pode receber o valor de 0 a 54, que correspondem ao *slots* das prateleiras);
- `Trigger_receive`: responsável por receber o sinal de gatilho para a execução do serviço de receber, o qual comanda os atuadores do transelevador de forma a pegar um caixote numa esteira de destino;
- `Trigger_insert`: responsável por receber o sinal de gatilho para a execução do serviço de inserir, o qual comanda os atuadores do transelevador de forma a inserir um caixote numa posição da prateleira;
- `At_left`: recebe o sinal de confirmação de posição à esquerda do atuador responsável por inserir e pegar o caixote, o qual precisa ir à esquerda (em direção à esteira de destino) para fazer a pega;
- `At_right`: recebe o sinal de confirmação de posição à direita do atuador responsável por inserir e pegar o caixote, o qual precisa ir à direita (em direção à prateleira) para fazer a inserção;
- `At_middle`: recebe o sinal de confirmação de posição no meio do atuador responsável por inserir e pegar o caixote, o qual precisa ir ao meio (em direção à prateleira) para voltar depois de pegar o caixote;
- `Is_moving_X`: recebe o sinal de confirmação de o transelevador está movendo em X, utilizado para saber quando o mesmo encerrou um movimento;

- `Is_moving_Z` : recebe o sinal de confirmação de o transelevador está movendo em Z, utilizado para saber quando o mesmo encerrou um movimento;

Quanto às saídas, tem-se:

- `Left` : saída que controla o atuador do transelevador capaz de ir à esquerda para fazer pega;
- `Right` : saída que controla o atuador do transelevador capaz de ir à direita para fazer a inserção;
- `Lift` : saída que controla o atuador do transelevador capaz de levantar o caixote, de forma a retirar da esteira de destino;
- `Target_position` : saída que controla o movimento do transelevador - para o valor 0, o mesmo fica parado, enquanto que para os demais valores até 54, ele se movimenta até a posição especificada da prateleira;
- `Moved` : saída de confirmação da finalização do serviço de mover para uma determinada posição;
- `Inserted` : saída de confirmação da finalização do serviço de inserção do caixote;
- `Received` : saída de confirmação da finalização do serviço de receber (ou pegar) o caixote.

5.6 Classes dos agentes

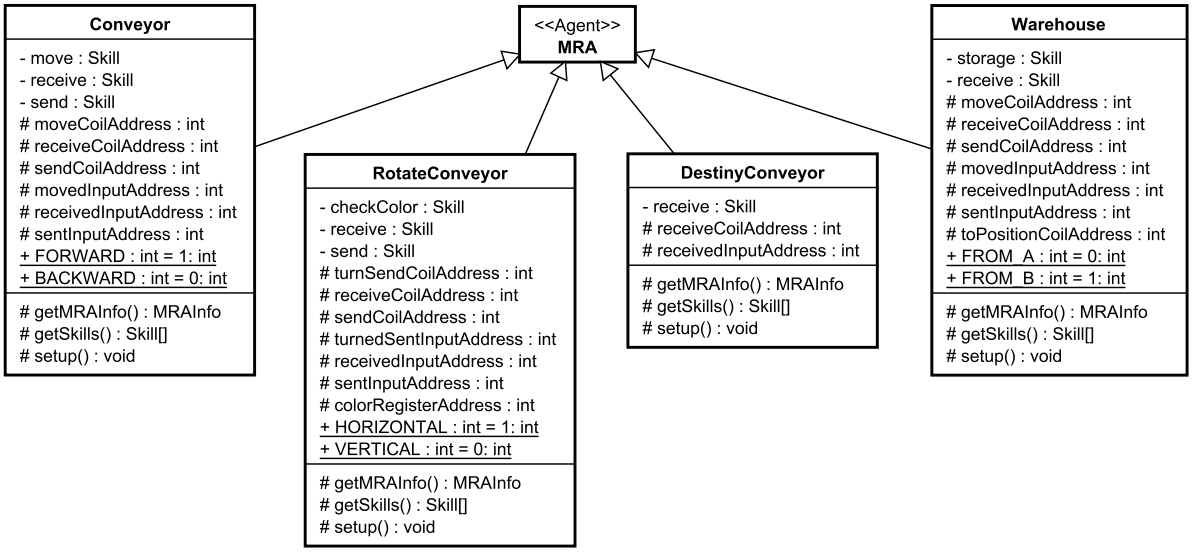
Como dito anteriormente, a arquitetura proporciona várias classes que servem de base para uma modelagem MAS. Cabe ao aplicador, então, usar tais classes a sua disposição, assim como criar novas classes (específicas para aplicação) que as utilizem. Para a descrição do Staudinger, seguindo as divisões de camadas, tem-se as seguintes classes.

- Camada física:
  - `Conveyor` : classe filha de `MRA` que modela os módulos Conveyor Belt e tem a skill `move` , a qual move o caixote para a posição solicitada ligando o motor da esteira;
  - `RotateConveyor` : classe filha de `MRA` que modela os módulos Rotate Conveyor Belt e conta com três skills: `receive` , `checkColor` e `move` . A primeira é responsável por rotacionar a esteira até a posição solicitada e ligar o motor, de modo a receber um caixote dessa direção. A segunda verifica se a cor do caixote coincide

com a cor passada como parâmetro, se sim, retorna `true`. A última move o caixote para a posição solicitada utilizando a esteira do módulo;

– `DestinyConveyor`: classe filha de `MRA` responsável por modelar as esteiras de acesso ao armazém. Conta com a skill `receive`, que é responsável por ligar o motor da esteira de modo a receber um caixote de uma direção especificada através dos argumentos passados.

Figura 35 – Classes da camada física.



Fonte: De autoria própria.

- Camada cognitiva:

  – `NewOrder`: modela uma nova produção até que o caixote tenha sua cor identificada. Ao ser instanciada, recebe dois parâmetros: a quantidade de bolinhas e a cor do produto. No seu método `produce()` são chamadas as skills da `StorageConveyor` e da primeira `RotateConveyor`, a fim de pegar um novo caixote, verificar sua cor e decidir se o caixote será utilizado numa nova produção ou inserção. A partir disso, o agente instância dessa classe solicita uma instanciação de um novo agente do tipo `Production` ou `Insert` ao agente Instanciador;

  – `Insert`: é uma classe filha de `Product` que representa uma nova inserção. Ao ser instanciada, recebe a cor do caixote como parâmetro. Tal inserção é feita após um caixote de cor indesejada ser retirado e analisado, de forma que seja guardado no armazém para uma produção futura. Para isso, no método `produce()`, o agente do tipo `Insert` faz as chamadas de skill necessárias para levar o caixote até a esteira de destino adequada;

  - `Production`: é uma classe filha de `Product` que representa uma produção de fato. Ao ser instanciada, recebe a quantidade de bolinhas do produto como parâmetro. Tal produção é feita após um caixote da cor requisitada ser retirado e analisado, de forma que o mesmo seja imediatamente utilizado na produção. Para isso, no método `produce()`, o agente do tipo `Production` faz as chamadas de skills necessárias para levar o caixote para receber bolinhas, ser tampado e chegar até à esteira de destino adequada.

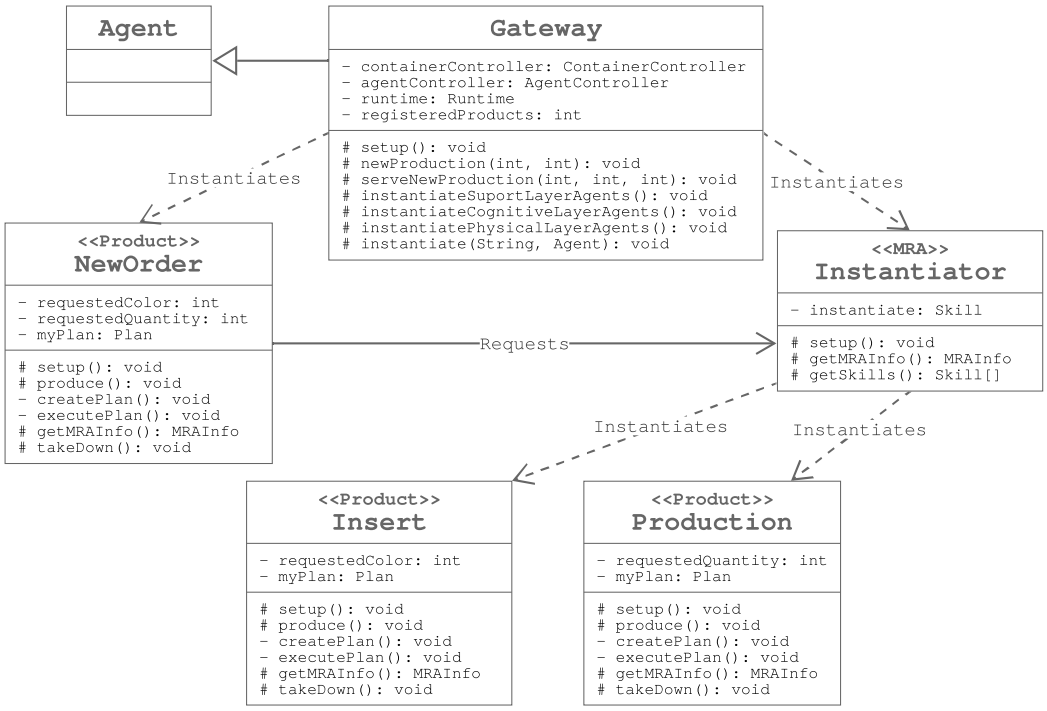
- Camada de suporte:

  - `Instantiator`: classe que descreve um agente instanciador. É um agente `MRA` da camada cognitiva responsável por instanciar outros agentes para os agentes do tipo `Product` através da skill `instantiate`, a qual recebe uma `String` para identificar o nome do agente a ser instanciado, e outra para identificar de qual classe será tal agente;
  - `YPA`: classe responsável por implementar um agente com o papel de páginas amarelas na arquitetura (Yellow Page Agent). Tal classe possui métodos que implementam os serviços de busca por skill, registro e desregistro de agentes numa tabela de itens do tipo `MRAInfo`.

- Camada de negócios:

  - `Gateway`: é a classe responsável por representar o agente Gateway da arquitetura, que, por sua vez, encabeça o sistema como um todo e instancia os agentes das camadas mais baixas. Por isso, conta com métodos que encapsulam toda a produção, nos quais são instanciados agentes cognitivos para os quais delegam-se distribuidamente as tarefas. Conta também com uma lista de pedidos na qual o método `addNewRequest()` adiciona um novo pedido de produção. Tal método recebe como parâmetro a cor do caixote. A execução da lista de pedidos é feita pelo método `executeRequests()`.

Figura 36 – Classes da camada cognitiva e de negócios.



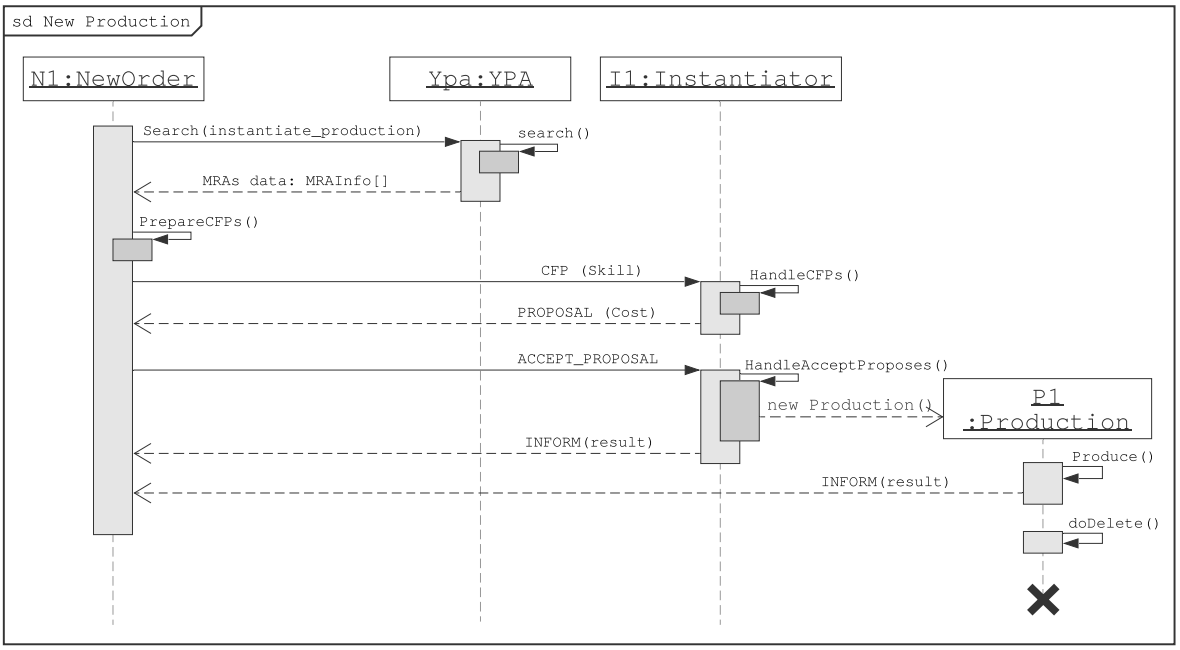
Fonte: De autoria própria.

5.7 Funcionamento geral

O intuito da descrição do Staudinger é oferecer ao usuário final o serviço de produção de um ou mais produtos em série. Por isso, o agente que encabeça o sistema (Gateway) conta com métodos que encapsulam toda a produção e proporciona tal serviço. O usuário, portanto, poderá, especificar a cor e, em seguida, adicionar tal pedido na lista (`addNewRequest()`). Uma vez que tal lista esteja completa, o usuário poderá pedir para o sistema começar a produzir (`executeRequests()`).

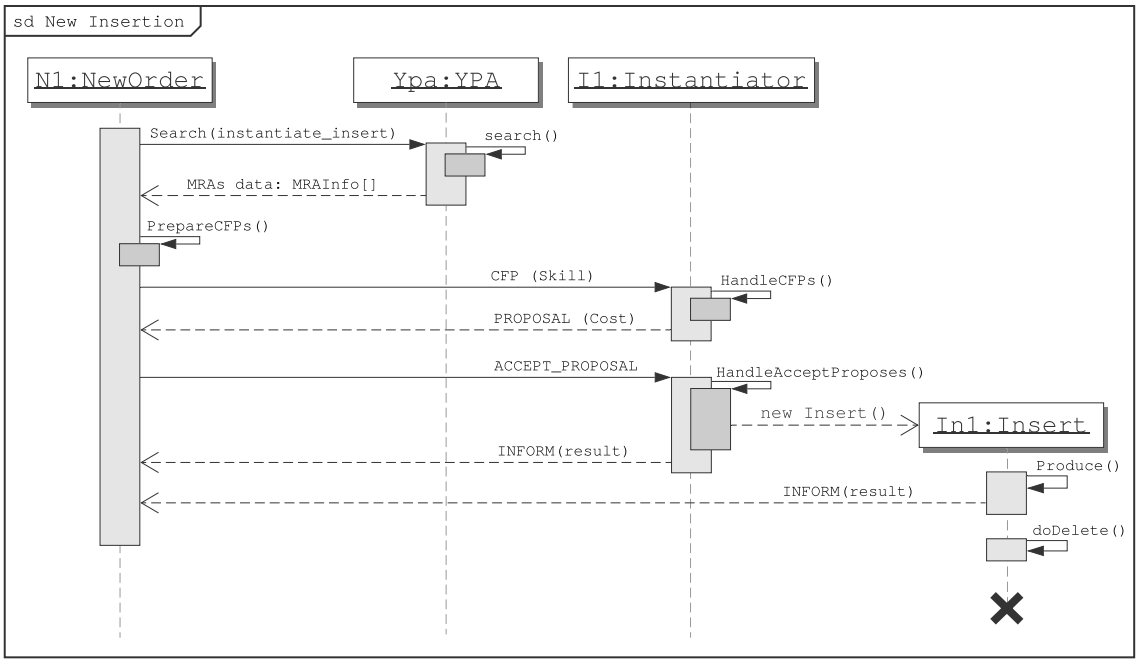
Uma vez que for solicitado a produção, o agente `Gateway` irá instanciar um agente da classe `NewOrder`, para o qual uma tentativa de produção será delegada. Tal agente será responsável por levar o caixote da pilha até à esteira rotativa, onde será checado sua cor. Caso a cor coincida com a desejada para o pedido, o agente `NewOrder` solicitará a instanciação de um agente do tipo `Production` para o `Instantiator`. Caso contrário, a solicitação de instanciação será de um agente do tipo `Insert`.

Figura 37 – Sequência para uma nova produção.



Fonte: De autoria própria.

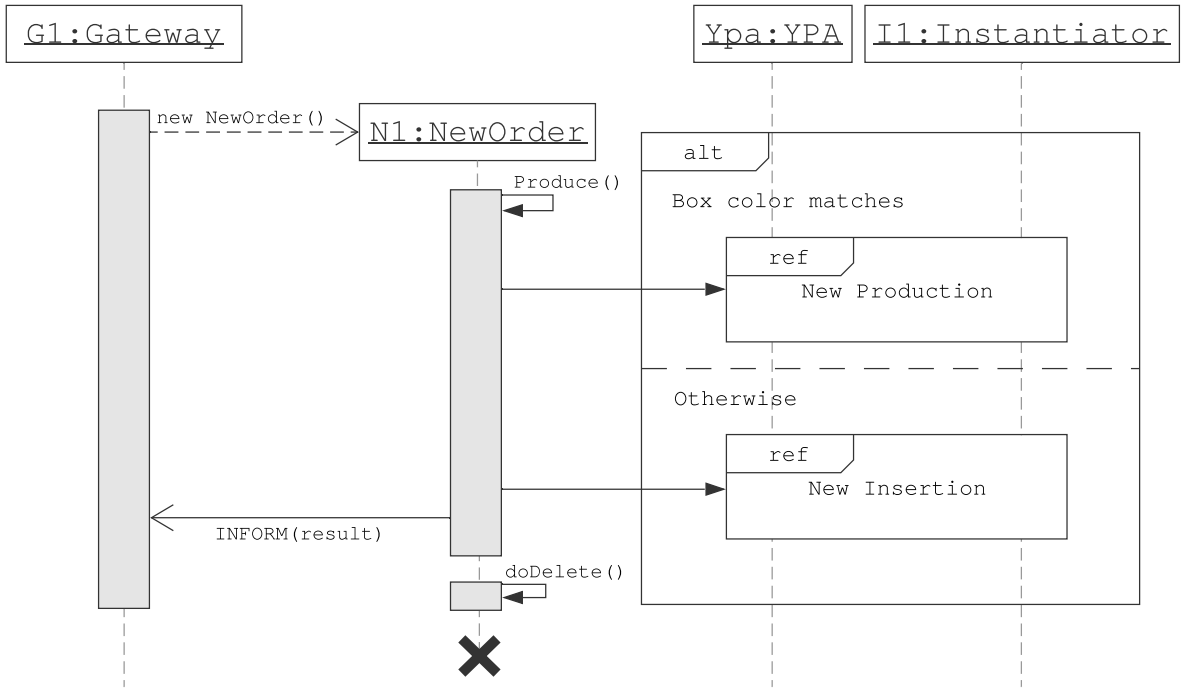
Figura 38 – Sequência para uma nova inserção.



Fonte: De autoria própria.

Uma vez que tais agentes sejam instanciados, eles serão responsáveis por levar o caixote ao destino adequado. Como comentado anteriormente, tais processos são feitos com um plano de execução, onde chamadas remotas de skills de agentes da camada física são feitas.

Figura 39 – Sequência para uma nova tentativa produção.



Fonte: De autoria própria.

Caso a tentativa de produção não dê certo, isto é, caso a cor do caixote não coincida, o agente `Gateway` irá tentar novamente repetindo o processo, ou seja, instanciando um novo agente `NewOrder`. Isso acontecerá indefinidamente até que a cor coincida e o pedido seja realizado.

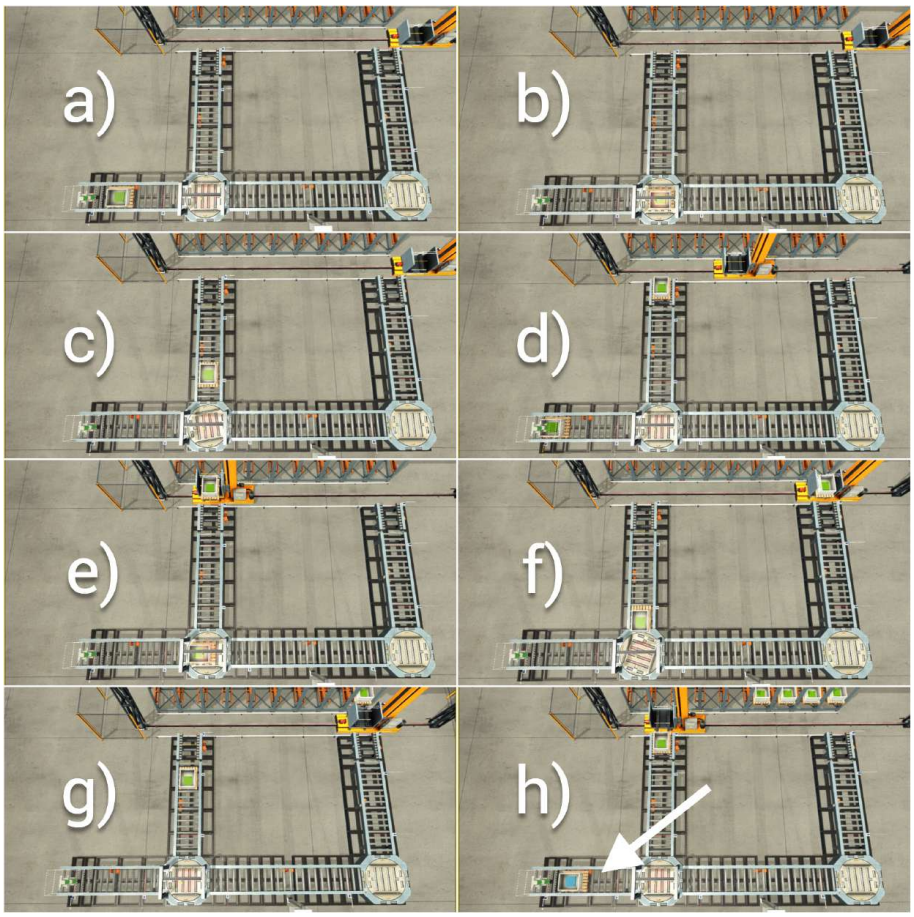
6 Resultados e Discussão

Esta seção tem como objetivo mostrar os resultados da descrição do comportamento adotado para o sistema através dos registros de interação entre os agentes, que são a saída em tempo real sistema multiagente do código Java, ao passo que ilustra como o processo se desdobra na planta 3D.

Primeiramente, apresenta-se a visão geral de uma tentativa de produção, isto é, a inserção de caixotes de cor não desejada até que seja emitido um caixote com a cor solicitada. Em seguida, apresenta-se uma visão mais detalhada do processo de inserção e produção, comparando com os registros do sistema multiagente Java.

6.1 Visão geral do funcionamento

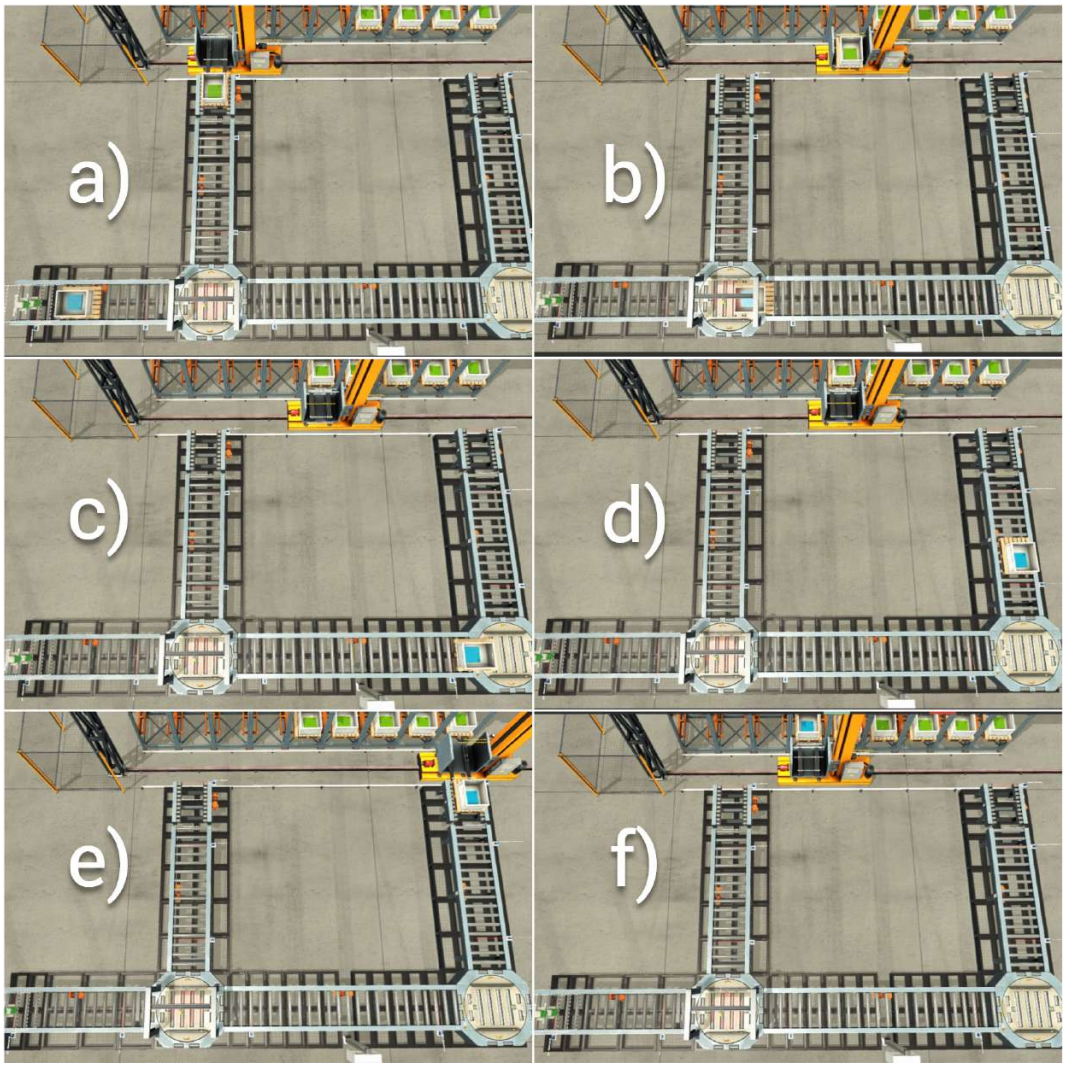
Figura 40 – Visualização 3D de tentativas de produção de um caixote azul.



Fonte: De autoria própria.

A Figura 40 apresenta o comportamento do sistema ao tentar produzir um produto da cor azul e ter de lidar com várias tentativas nas quais a cor não coincidia. Portanto, observa-se que os caixotes com conteúdo verde (cor não desejada) são inseridos (processo de inserção), enquanto novos caixotes são emitidos a fim de tentar novamente. Na Figura 40-h é possível ver que é emitido, finalmente, um caixote com a cor desejada, depois de 4 tentativas sem sucesso. A Figura 41, por sua vez, mostra a produção do produto, isto é, o caixote da cor solicitada seguindo o caminho da produção.

Figura 41 – Visualização 3D da produção de um caixote azul.



Fonte: De autoria própria.

6.2 Simulação da planta virtual

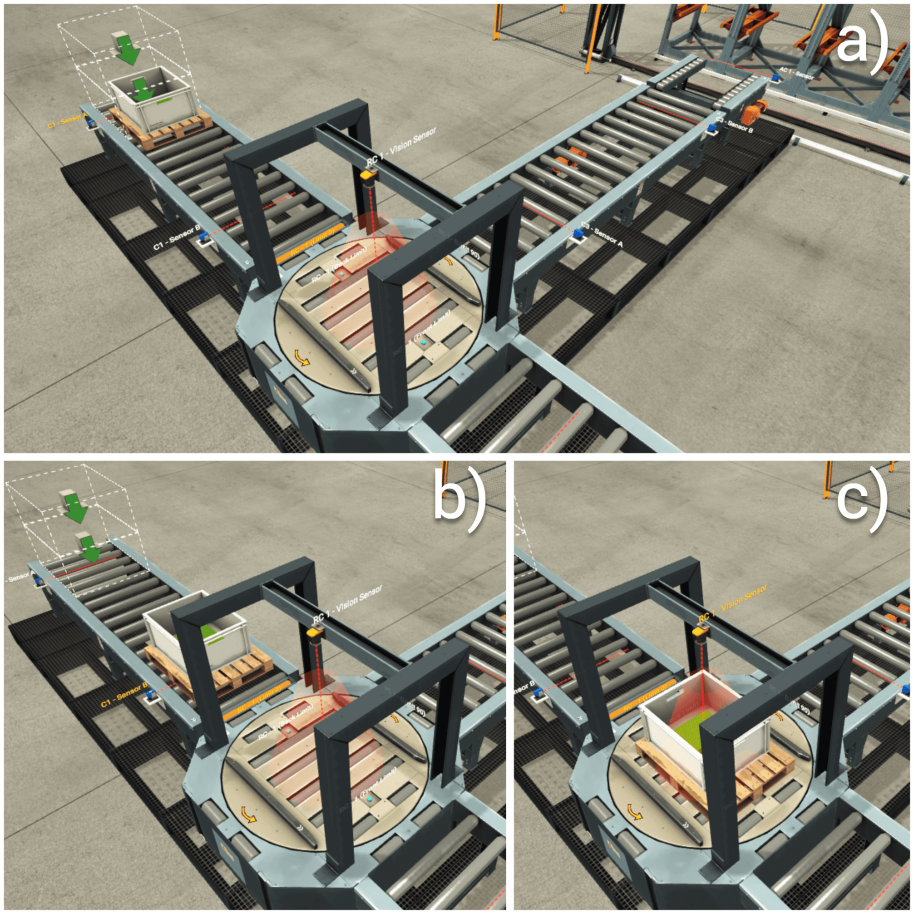
Como a descrição é extensa e o processo é bem ilustrado pelas imagens da planta, optou-se por mostrar a simulação para o caso de um único produto solicitado. Esse produto é de cor azul e, durante a simulação, não coincidentemente, o primeiro caixote é de cor verde e o seguinte

é azul, de forma a ilustrar o processo de inserção e produção.

O primeiro caixote a ser emitido aconteceu de ser um caixote verde, ou seja, um caixote cuja cor não é a desejada. Assim, pode-se observar um exemplo de inserção de caixote, uma vez que não é possível usar o mesmo na produção solicitada. A Figura 42 mostra em sequência o caixote: após ser emitido (Figura 42-a); após ser movido pela esteira de p1 para p2 (Figura 42-b); e após ser enviado pela esteira 1 e recebido pela esteira rotativa 1, indo de p2 para p3 (Figura 42-c).

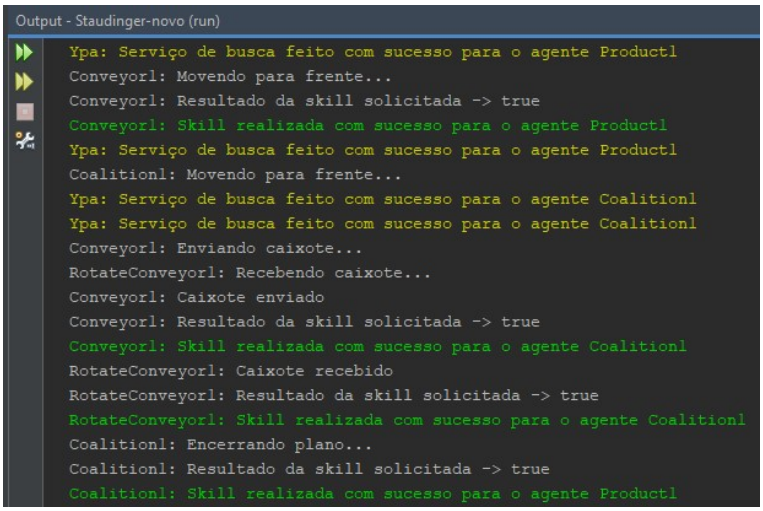
A Figura 43, por sua vez, mostra a comunicação entre os agentes para o processo mostrado na Figura 42: o agente produto 1 procura através do YPA a skill de mover de p1 para p2; após receber a resposta, ele solicita ao agente esteira 1 e é atendido; procura através do YPA a skill de mover de p2 para p3; após receber a resposta, ele solicita ao agente de coalisão 1, o qual se encarrega de solicitar ao YPA pelos agentes capazes de receber/enviar nas posições especificadas; por fim, a esteira 1 e a esteira rotativa 1 movem de p2 para p3, de forma a atender o pedido do coalisão 1.

Figura 42 – Visualização 3D do início de uma tentativa de produção - caixote verde.



Fonte: De autoria própria.

Figura 43 – Registro de saída do programa Java de uma tentativa de produção - caixote verde.



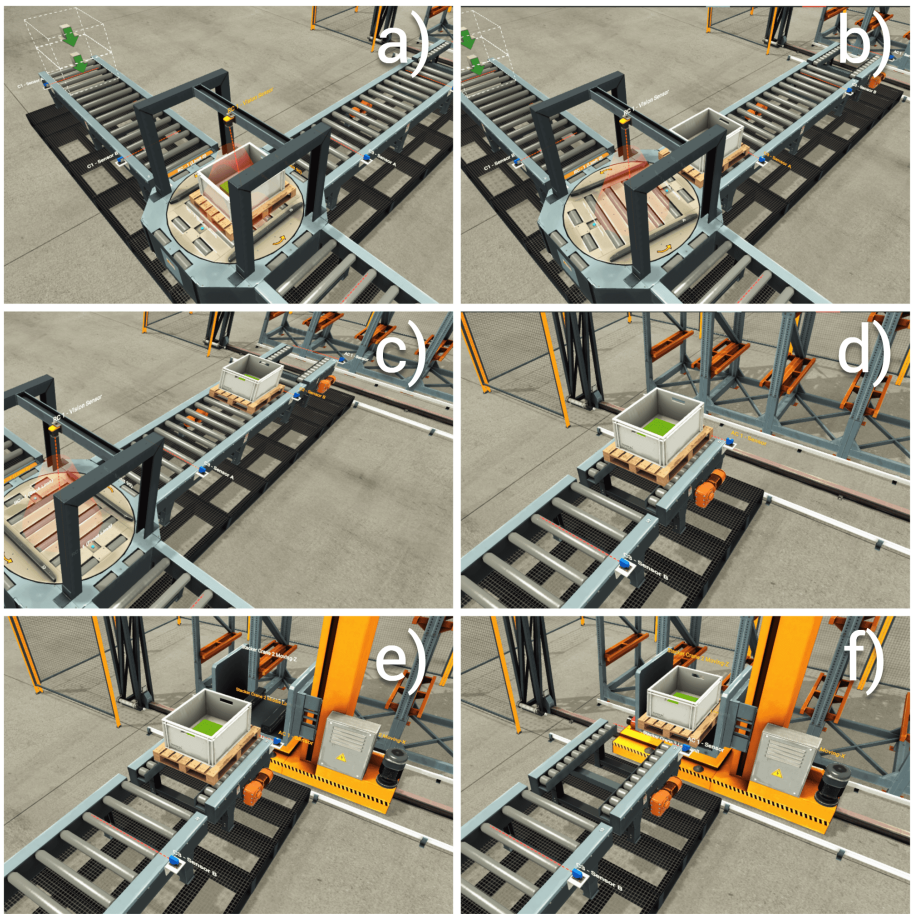
Fonte: De autoria própria.

Após ser recebido pela esteira rotativa 1, o caixote em questão tem a cor identificada pelo sensor de cor presente no módulo. Como a cor não é a desejada na produção solicitada, é tomada a decisão de seguir pela rota A, isto é, o caminho de inserção.

A Figura 44 mostra tal processo, onde o caixote: é recebido pela esteira rotativa 1 para a checagem de cor (Figura 44-a); é enviado de p3 para p7 pela esteira rotativa 1 e esteira 3 (Figura 44-b); é movido de p7 para p8 pela esteira 3 (Figura 44-c); é movido de p8 para p9, cuja posição final é a esteira de destino 1 (Figura 44-d); e recebido pelo armazém, o qual move até a posição A (Figura 44-e); e recebe (ou pega) o caixote na esteira de destino 1 (Figura 44-f).

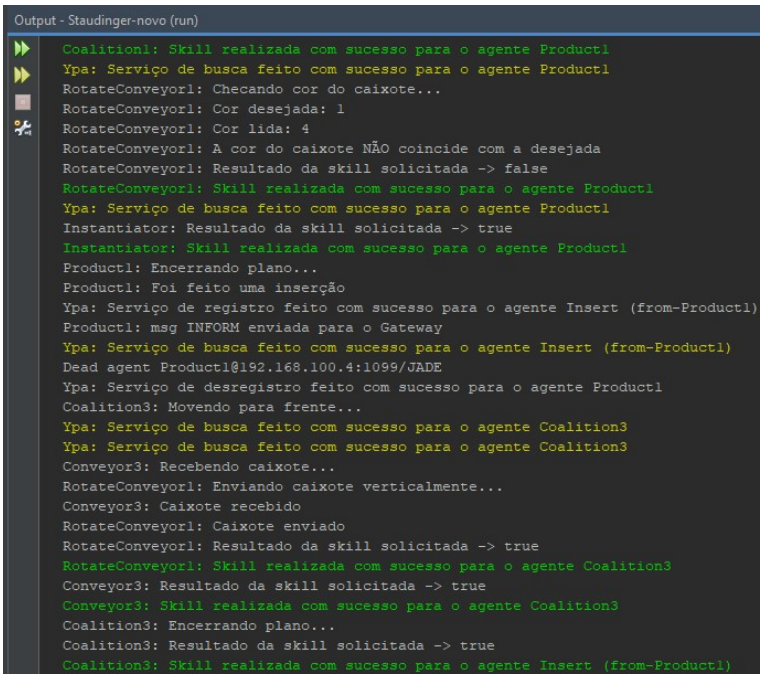
A Figura 45 mostra os registros do sistema multiagente para a sequência da Figura 44. Como é possível observar, é solicitado ao YPA a skill de checagem de cor, a qual teve como resultado o valor booleano falso, implicando que a cor não coincide com a desejada. Em seguida, é solicitado ao agente instanciador a instanciação de um agente de inserção, o qual encapsula todo o processo de inserção. Por fim, o agente inserção solicita ao agente de coalisão a execução da skill responsável por passar o caixote da esteira rotativa 1 para a esteira 3.

Figura 44 – Visualização 3D de uma inserção - caixote verde.



Fonte: De autoria própria.

Figura 45 – Registro de saída do programa Java de uma inserção - 1/3.

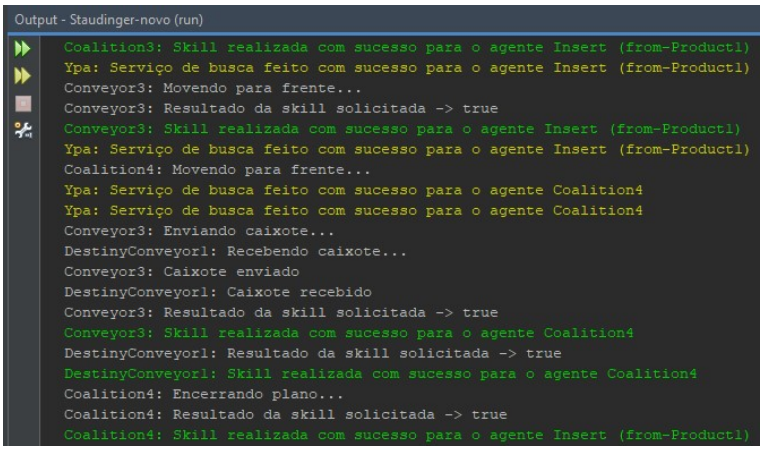


Fonte: De autoria própria.

Já na Figura 46, observa-se a continuação dos registros do processo da Figura 44, ou seja, o caixote sendo movido até a esteira de destino 1, para, em seguida, ser recebido pelo armazém, cujo registro de comunicação dos agentes é mostrado na Figura 47. Nela pode ser visto o agente armazém executando a skill de receber (de A), mas, ao mesmo tempo, é registrado também o início de uma nova tentativa de produção, uma vez que a tentativa atual falhou (virou inserção). Por isso (em azul), é visto o agente Gateway registrando que está sendo instanciado um novo agente produção, cujo o nome recebe a identificação do produto (1) e a tentativa (2): "Production1_2".

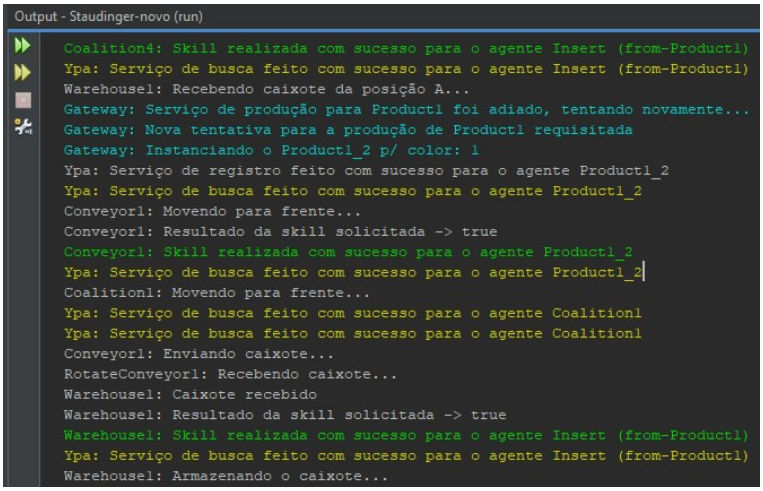
É importante notar também, na última linha da Figura 47, o início da execução da skill de armazenar, pelo agente armazém. É mostrado somente o início pois essa skill demora ser executada e agora um novo caixote surgiu no sistema (nova tentativa de produção), logo, diversas outras skills aparecerão no registro da comunicação e a skill pode ser vista sendo finalizada somente posteriormente (Figura 50).

Figura 46 – Registro de saída do programa Java de uma inserção - 2/3.



Fonte: De autoria própria.

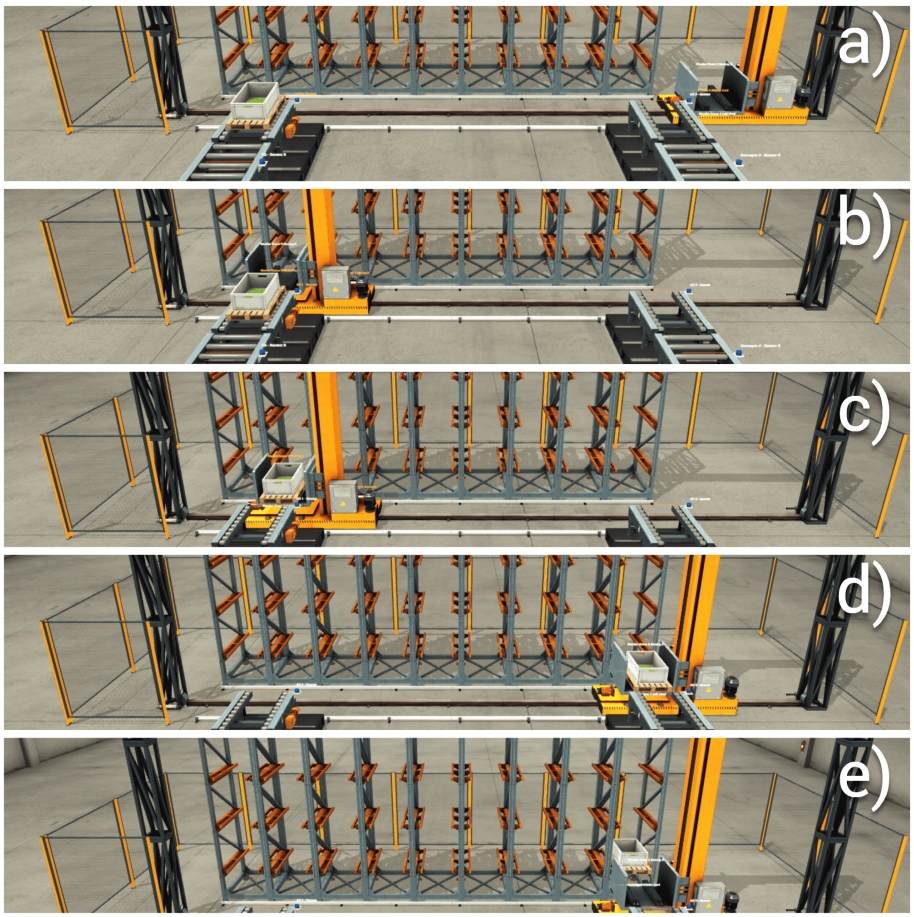
Figura 47 – Registro de saída do programa Java de uma inserção - 3/3.



Fonte: De autoria própria.

O processo de armazenamento de um caixote (da posição A) é mostrado na Figura 48, seguindo o seguinte fluxo: o receptor móvel do armazém (transelevador) move-se para a posição da esteira de destino 1 (Figura 48-b); pega o caixote (recebe) (Figura 48-c); move-se para a posição de inserção (Figura 48-d), que é, nesse caso, a posição 1, pois é o primeiro caixote a ser armazenado; e insere no armazém vertical (Figura 48-e).

Figura 48 – Visualização 3D do armazenamento de A - inserção.



Fonte: De autoria própria.

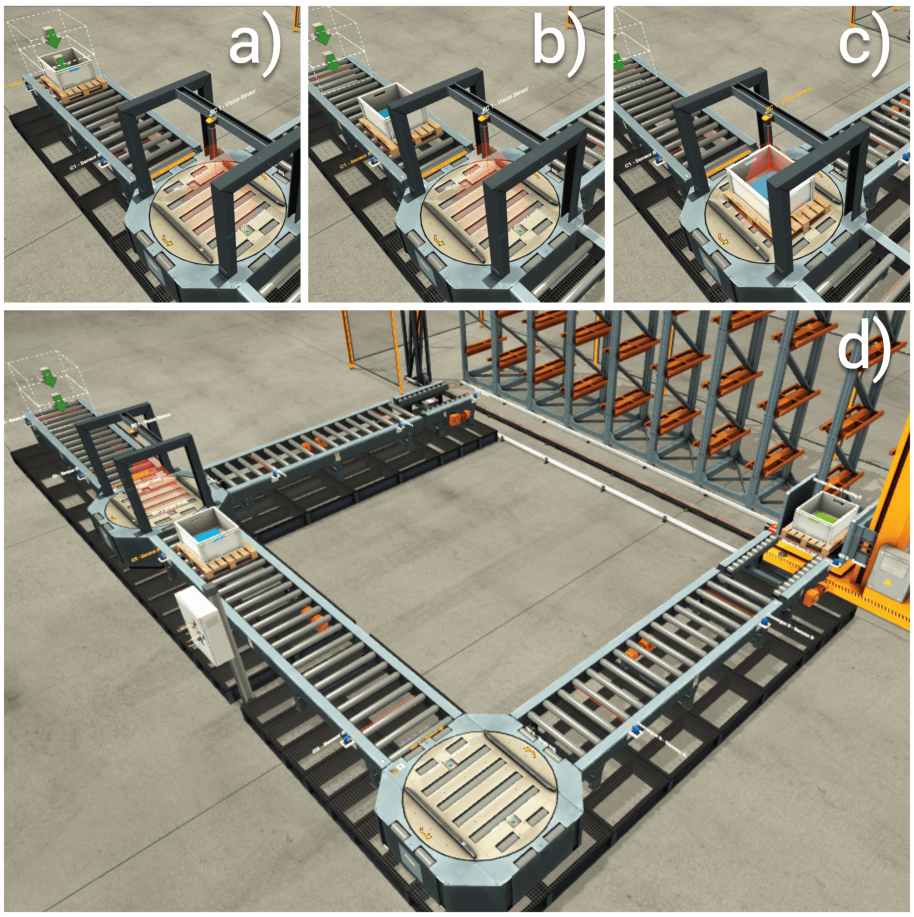
O segundo caixote aconteceu de ser da cor azul, que coincide com a solicitada. Assim, pode-se observar um exemplo de produção, a qual é mostrada na Figura 49, onde observa-se, em sequência, o caixote: após ser emitido (Figura 49-a); após ser movido pela esteira de p1 para p2 (Figura 49-b); e após ser enviado pela esteira 1 e recebido pela esteira rotativa 1, indo de p2 para p3 (Figura 49-c).

Após ser recebido pela esteira rotativa 1, é feito a identificação da cor e, em seguida, o caixote é enviado para a esteira 2, a qual o recebe (Figura 49-d). A Figura 50 mostra o registro do agente produto que representa a segunda tentativa de produção e sua comunicação com os agentes cyber-físicos e de coalisão para esse processo inicial de levar o caixote até à esteira rotativa para checagem de cor.

Identificou-se, portanto, a cor desejada e seguiu-se para a produção, como pode ser visto na Figura 51, onde é visto o agente da esteira rotativa executando a skill de checagem de cor com valor desejado e lido iguais, o que implicou no resultado booleano verdadeiro para a execução da skill. Em seguida, ainda na figura mencionada, é visto a solicitação de instanciação de um novo agente do tipo produção, o qual encapsula o processo de produção e solicita o movimento

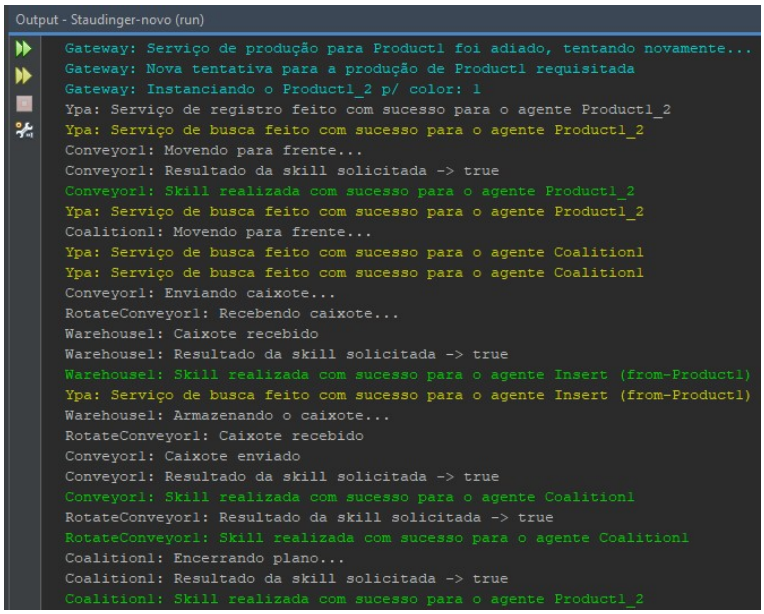
de p3 para p4 ao agente de coalisão 2.

Figura 49 – Visualização 3D do início de uma tentativa de produção - caixote azul.



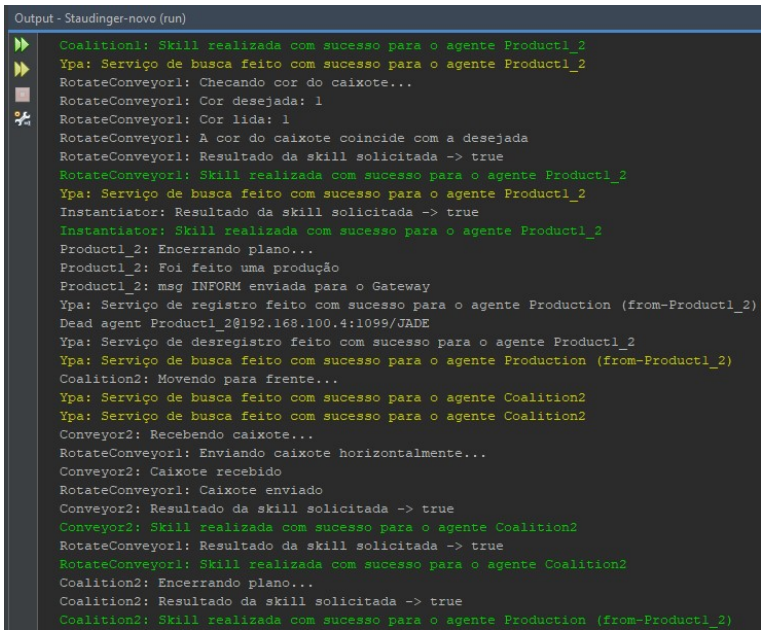
Fonte: De autoria própria.

Figura 50 – Registro de saída do programa Java de uma tentativa de produção - caixote azul.



Fonte: De autoria própria.

Figura 51 – Registro de saída do programa Java de uma produção - 1/4.

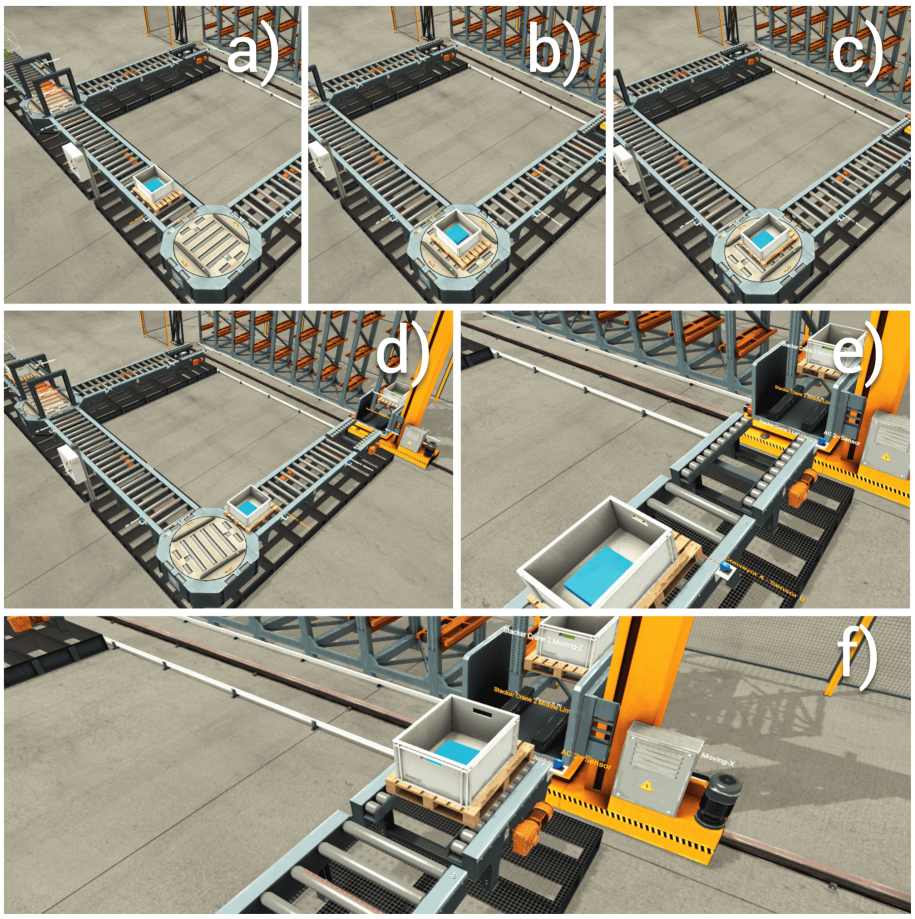


Fonte: De autoria própria.

Em seguida, como nota-se na Figura 52, o caixote segue a rota B (isto é, a rota de produção), sendo movido: de p4 para p5 pela esteira 2 (Figura 52-a); de p5 para p6 pela esteira 2 e esteira rotativa 2 (Figura 52-b); de p6 para p10 pela esteira rotativa 2 e esteira 4 (Figura 52-c e 52-d); de p10 para p11 pela esteira 4 (Figura 52-e); e, finalmente, de p11 para p2 pela esteira 4 e esteira de destino 2 (Figura 52-f).

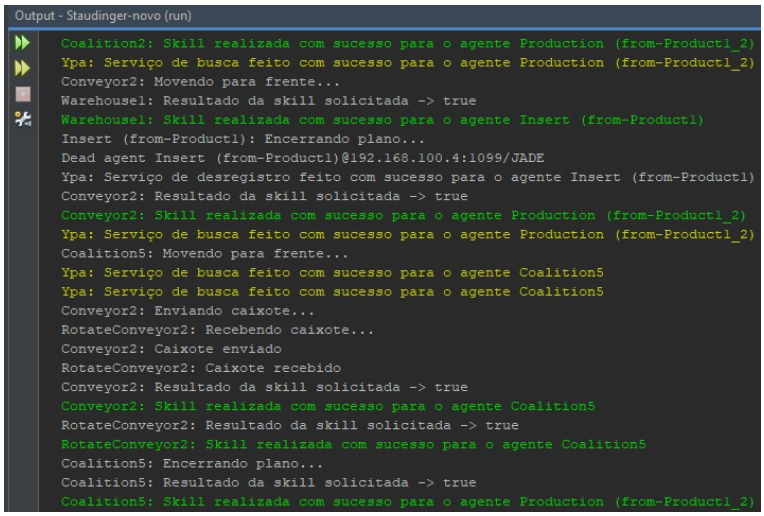
Já na Figura 53, 54 e (55), observa-se as interações entre os agentes para o processo de produção. Em (53) o agente esteira 2 move o caixote e o envia para a esteira rotativa 2, a qual o recebe. Já em (54), o agente esteira rotativa 2 envia o caixote ao agente esteira 4, o qual recebe e o move para p11. Por fim, em (55) o caixote é enviado pelo agente esteira 4 e recebido pelo agente esteira de destino 2.

Figura 52 – Visualização 3D de uma produção - caixote azul.



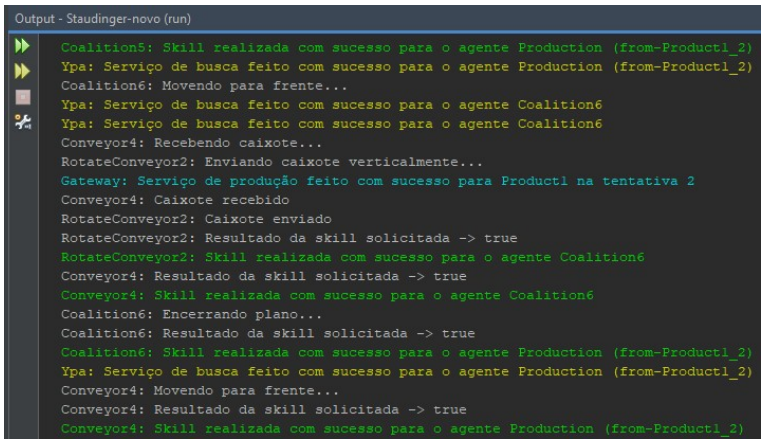
Fonte: De autoria própria.

Figura 53 – Registro de saída do programa Java de uma produção - 2/4.



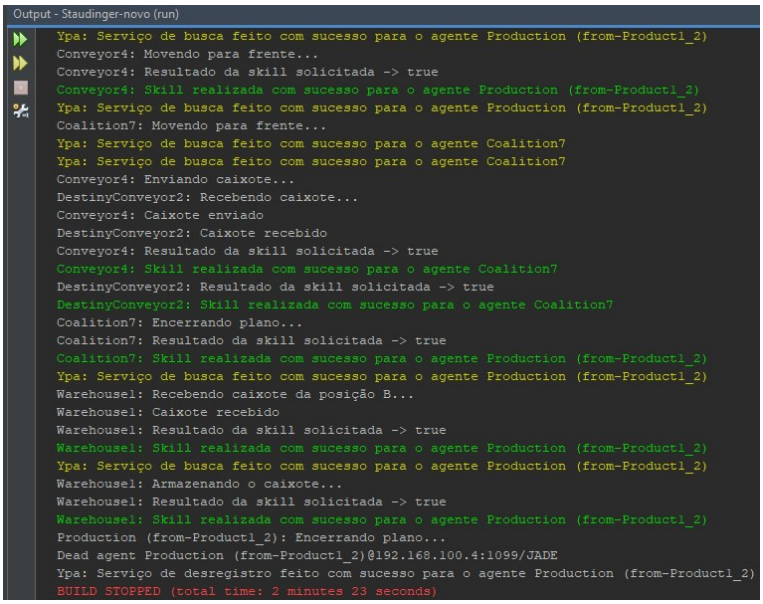
Fonte: De autoria própria.

Figura 54 – Registro de saída do programa Java de uma produção - 3/4.



Fonte: De autoria própria.

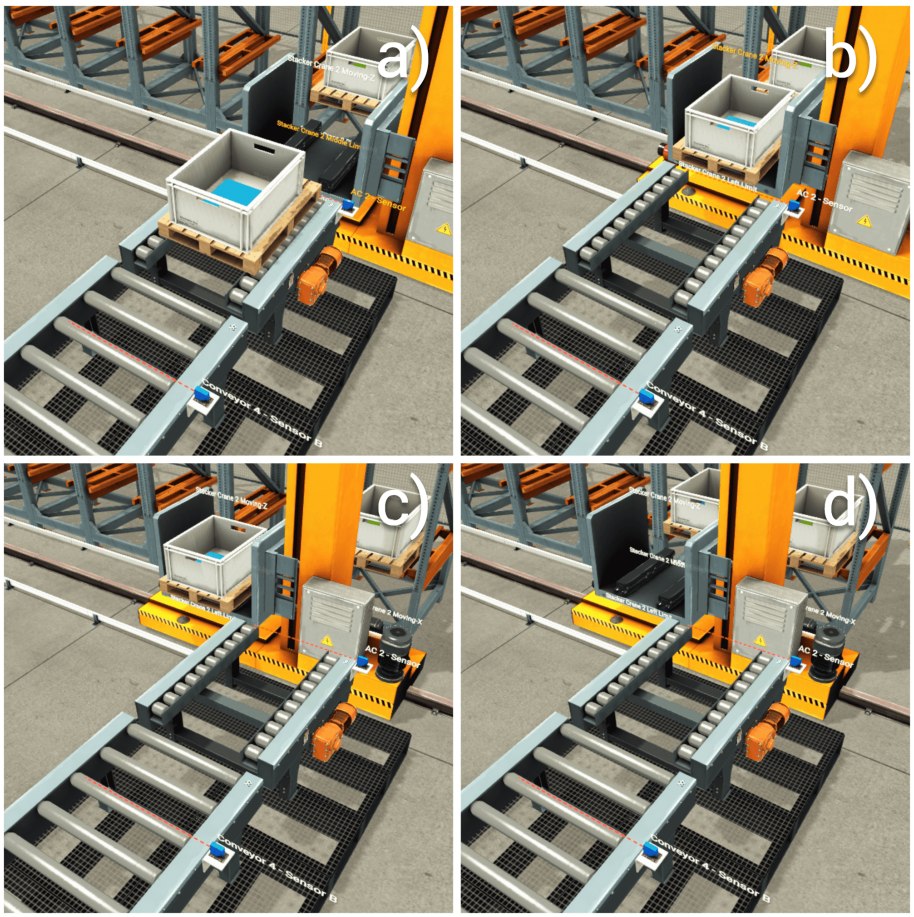
Figura 55 – Registro de saída do programa Java de uma produção - 4/4.



Fonte: De autoria própria.

Ainda na Figura 55, observa-se o processo de recebimento e armazenamento por parte do agente armazém, que agora o faz da posição B e insere o caixote azul na posição 2, conforme mostrado na Figura 56. O fluxo segue da seguinte forma: o receptor móvel do armazém (transelevador) não move-se para a posição da esteira de destino 2, pois ele já estava nessa posição (Figura 48-a); pega o caixote (recebe) (Figura 48-b); move-se para a posição de inserção (Figura 48-c), que é, nesse caso, a posição 2, pois é o segundo caixote a ser armazenado; e insere no armazém vertical (Figura 48-d).

Figura 56 – Visualização 3D do armazenamento de B - inserção.



Fonte: De autoria própria.

7 Considerações Finais

Foram feitos estudos teóricos a respeito de sistemas evolutivos de produção, sistemas multiagentes e do *framework* JADE, o demonstrador de manufatura Staudinger foi descrito e o comportamento adotado foi alcançado com um sistema multiagente que é capaz de auto-organização. Além disso, foi alcançado também uma forma de visualização 3D da planta em funcionamento em tempo real, o que permite simular e acompanhar o desenvolvimento de um sistema multiagente voltado para processo industrial utilizando a arquitetura EPSCore.

Para alcançar tais feitos, foram feitas algumas modificações na arquitetura, principalmente na classe do agente cyber-físico (chamada originalmente de `MRA`), a qual foi melhorada a fim de desenvolver melhor o conceito de agente cyber-físico. Para esse trabalho, essas mudanças incluem principalmente a adição de um cliente Modbus TCP/IP na classe em questão, a implementação de um agente de coalisão e o uso do CFP, que permite uma negociação com uma rede de agentes. As mudanças descritas acompanham, ainda, diagramas em UML.

No que diz respeito à visualização do sistema em 3D, destaca-se o estabelecimento da comunicação e da arquitetura proposta, a qual foi descrita em detalhes e validada com um estudo de caso. Tal estudo contemplou todo as camadas, isto é: o sistema multiagente, a programação Ladder/FBD, que foi desenvolvida de uma forma particular para um sistema multiagente, a montagem da planta 3D e as conexões entre as camadas. Ademais, todo o sistema foi simulado, se mostrou funcional e de fácil configuração com os softwares utilizados.

Portanto, o trabalho em questão, além de mostrar uma aplicação de um sistema multiagente voltado a um processo industrial, conclui que é possível realizar, de maneira fácil, o desenvolvimento e simulação de um sistema multiagente voltado para um processo industrial utilizando a pilha de softwares, protocolos e linguagens de programação citadas e da maneira descrita. Assim, pode-se obter um desenvolvimento mais maduro, uma vez que a simulação visualizada permite explorar possibilidades que somente o sistema físico final poderia oferecer, além de poder ser usada de forma didática no estudo de sistemas multiagentes para processos industriais.

Referências

ALSTERMAN, H.; ONORI, M. Definitions, limitations and approaches of evolvable assembly system platforms. In: SPRINGER. *International Conference on Information Technology for Balanced Automation Systems*. [S.l.], 2004. p. 367–377. Citado na página 13.

BELLIFEMINE, F.; CAIRE, G.; GREENWOOD, D. *Developing Multi-Agent Systems with Jade*. [S.l.]: Wiley, 2007. Citado na página 16.

CAIRE, G. Jade tutorial: Jade programming for beginners. *http://jade. tilab. com/doc/JADEProgrammingTutorial-for-beginners. pdf*, 2009. Citado 2 vezes nas páginas 18 e 19.

CAVALCANTE, A. L. D. *Arquitetura Baseada em Agentes e Auto-organizável para a Manufatura*. [S.l.]: Tese de Doutorado. UFRGS, Porto Alegre, 2012. Citado 5 vezes nas páginas 13, 16, 17, 18 e 21.

Factory IO. *Setting up CODESYS OPC UA (SP17 or higher)*. 2023. Disponível em: <https://docs.factoryio.com/tutorials/codesys/setting-up/codesys-opc-ua-sp17/>. Citado na página 41.

FOROUZAN, B. A. *Comunicação de dados e redes de computadores*. [S.l.]: AMGH Editora, 2009. Citado na página 25.

FREI, R. et al. An architecture for self-managing evolvable assembly systems. In: IEEE. *2009 IEEE International Conference on Systems, Man and Cybernetics*. [S.l.], 2009. p. 2707–2712. Citado na página 21.

HOSSEINPOUR, F.; HAJIHOSSEINI, H. Importance of simulation in manufacturing. *World Academy of Science, Engineering and Technology*, v. 51, n. 3, p. 292–295, 2009. Citado na página 14.

KARNOUSKOS, S. et al. Industrial agents as a key enabler for realizing industrial cyber-physical systems: Multiagent systems entering industry 4.0. *IEEE Industrial Electronics Magazine*, IEEE, v. 14, n. 3, p. 18–32, 2020. Citado na página 13.

LEE, E. Computing foundations and practice for cyber physical systems: A preliminary report. 01 2007. Citado na página 16.

LEITÃO, P.; KARNOUSKOS, S. Industrial agents: emerging applications of software agents in industry. Morgan Kaufmann, 2015. Citado na página 13.

LEITÃO, P. et al. Smart agents in industrial cyber–physical systems. *Proceedings of the IEEE*, v. 104, p. 1086–1101, 03 2016. Citado na página 16.

MCARTHUR, S. et al. Multi-agent systems for power engineering applications—part i: Concepts, approaches, and technical challenges. *Power Systems, IEEE Transactions on*, v. 22, p. 1743 – 1752, 12 2007. Citado na página 16.

MENDONÇA, R. d. S. *Plataforma Didática para Desenvolvimento de Sistemas Multiagente*. [S.l.]: Dissertação de Mestrado. Ufam, Manaus, 2016. Citado 6 vezes nas páginas 14, 21, 22, 23, 24 e 25.

MENDONÇA, R. d. S.; CAVALCANTE, A. L. D.; JUNIOR, V. F. de L. Epscore: A didactic open architecture of an evolvable production system. In: *2016 IEEE 21st International Conference on Emerging Technologies and Factory Automation (ETFA)*. [S.l.: s.n.], 2016. p. 1–4. ISSN null. Citado na página 21.

Modbus Organization. *MODBUS Application Protocol Specification: V1. 1b3*. [S.l.]: Modbus Organization, 2012. Citado na página 25.

ONORI, M.; BARATA, J.; FREI, R. Evolvable assembly systems basic principles. In: SPRINGER. *International Conference on Information Technology for Balanced Automation Systems*. [S.l.], 2006. p. 317–328. Citado na página 20.

OPC Foundation. *OPC 10000-1: UA Part 1: Overview and Concepts*. 2023. Disponível em: <https://reference.opcfoundation.org/Core/Part1/v105/docs/>. Citado na página 26.

SAKURADA, L.; LEITÃO, P. Multi-agent systems to implement industry 4.0 components. In: *2020 IEEE Conference on Industrial Cyberphysical Systems (ICPS)*. [S.l.: s.n.], 2020. v. 1, p. 21–26. Citado na página 13.

SINHA, R. et al. Modeling and simulation methods for design of engineering systems. *J. Comput. Inf. Sci. Eng.*, v. 1, n. 1, p. 84–91, 2001. Citado na página 14.

SUZIĆ, N. et al. Implementation guidelines for mass customization: current characteristics and suggestions for improvement. *Production Planning & Control*, Taylor & Francis, v. 29, n. 10, p. 856–871, 2018. Citado na página 14.

SWALES, A. et al. Open modbus/tcp specification. *Schneider Electric*, v. 29, p. 3–19, 1999. Citado na página 26.

WOLF, T. D.; HOLVOET, T. Emergence and self-organisation: a statement of similarities and differences. In: *Proceedings of the International Workshop on Engineering Self-Organising Applications 2004*. [S.l.: s.n.], 2004. p. 96–110. Citado na página 20.