UNIVERSIDADE FEDERAL DE SANTA CATARINA
CENTRO TECNOLÓGICO DE JOINVILLE
CURSO DE ENGENHARIA MECATRÔNICA

LUCAS DE CAMARGO SOUZA

A INDÚSTRIA 4.0 COMO REFERÊNCIA PARA MODELAGEM E IMPLEMENTAÇÃO DE COMPONENTES PLUG & PRODUCE EM UMA REDE INDUSTRIAL OPC UA

Joinville
2022

LUCAS DE CAMARGO SOUZA

A INDÚSTRIA 4.0 COMO REFERÊNCIA PARA MODELAGEM E IMPLEMENTAÇÃO DE COMPONENTES PLUG & PRODUCE EM UMA REDE INDUSTRIAL OPC UA

Trabalho de Conclusão de Curso apresentado como requisito parcial para obtenção do título de bacharel em Engenharia Mecatrônica no curso de Engenharia Mecatrônica, da Universidade Federal de Santa Catarina, Centro Tecnológico de Joinville.

Orientador: Dr. Giovani Gracioli

Joinville
2022

AGRADECIMENTOS

Ao Prof. Giovani Gracioli, agradeço por ter me apresentado o meu tópico de trabalho, cujo conhecimento faz um grande diferencial nas minhas experiências no meu ingresso no mundo profissional. Também agradeço ao Prof. Maurício Porath, também participante da banca, pela sua extraordinária influência no centro da UFSC de Joinville, mostrando aos estudantes o tamanho do mundo que eles são capazes de abraçar.

Especialmente à minha avó, Cida, dedico cada passo dado, cada fronteira que cruzo e a cada destino que alcanço, por causa dela, plantou-se o fruto do meu conhecimento e desenvolvimento como estudante, entusiasta do mundo e das razões das coisas. À minha família, meus pais Andréia e Roberto, meu irmão Leonardo, devo uma enorme retribuição por aquilo que me deram suporte, para eu chegar até a última palavra escrita deste trabalho, com força e motivação para seguir bravamente em frente. Também ao meu fiel e amigo companheiro, Angus, que foi minha zona de conforto durante o período de isolamento social causado pela pandemia da Covid-19.

À família Rosenkranz, que considero de coração e que me introduziu a Alemanha, dirijo as próximas palavras. *Es könnte nicht so perfekt gewesen sein, wäre es nicht die herzliche Unterstützung meiner lieben deutschen Familie aus Derschen. Mit ihnen habe ich gelernt, wie wertvoll das Leben sein kann, wenn wir uns austauschen dürfen. Sie haben mir die Tür der Welt eröffnet und ich lade sie ein, mit mir durch diese Tür zu gehen. Einen herzlichen Dank an meine deutsche Eltern, Inge und Johannes Rosenkranz. Ihr seid fantastisch.*

Tão especiais foram os meus amigos, que me surpreendem ao me mostrar o melhor da vida, o melhor contato, os melhores dias e noites, tão importante estando ao lado deles, os grupos do Capiroto Rotatório Mortal, que cresceu junto comigo, em Assis, e O Bolah, onde meu coração reside em Joinville, esses dois, especiais, a prova de quê fui capaz de encontrar as conexões perfeitas na minha vida.

É impossível escrever nessa página o número de tantas outras pessoas, em todo o mundo, que colaboraram não apenas com o meu desenvolvimento como estudante, mas com todos os meus colegas, que surpreenderam, inclusive a eles mesmos, ao reconhecer do que são capazes devido ao trabalho de profissionais que colaboram continuamente para a evolução da ciência e da tecnologia. Inúmeras outras instituições fizeram parte dessa jornada e poderiam ter seus nomes listados nesses agradecimentos, pois cada linha escrita vem das minhas anteriores vivências promovidas por pessoas dispostas a fazer a diferença.

Obrigado, pessoas, espero que nós sigamos em frente preocupando-nos com aquilo que há mais de importante para progredirmos, com nós mesmos, pessoas, todas e todos.

"Apaixone-se por alguma atividade, e a faça! Ninguém nunca descobre sobre o quê que é a vida, e isso nem importa. Explore o mundo. Quase tudo é realmente interessante se você for fundo o suficiente [...] não pense sobre quem você quer tornar, mas o que você quer fazer" (FEYNMAN, Richard P.; 1918-1988).

RESUMO

O modelo de referência de arquitetura da Indústria 4.0 recomenda o uso do padrão de comunicação OPC UA, que estabelece técnicas de representação virtual de componentes com base na abstração e descrição de suas funcionalidades. Um dos casos de uso é denominado fábricas adaptáveis, aquelas que dispensam a necessidade de reconfiguração dos processos ao introduzir ou modificar componentes na produção. Nesse contexto, este trabalho aborda a modelagem e implementação de componentes Plug & Produce interoperáveis e autônomos como solução para um sistema de produção auto-reconfigurável. O trabalho revisa as normas e técnicas de modelagem de componentes OPC UA e implementa um sistema demonstrativo caracterizado pela detecção de marcadores fiduciais utilizando dispositivos de câmera, componentes de processamento de imagem e composição automatizada das funcionalidades dos dispositivos, cuja abordagem é descrevê-las semanticamente como *skills*. O experimento mostra que tal sistema é interoperável e possui autonomia ao ser executado em diferentes servidores, contudo, mostra-se que, a partir de uma análise de uso de rede e de recursos operacionais, ainda carece de desenvolvimento para que atenda aos requisitos de sistemas distribuídos. Conclui-se que, apesar da complexidade resultante do uso de uma linguagem de baixo nível, a abordagem de descrição semântica de *skills* é eficaz e extensível para sistemas de produção autoadaptativos.

Palavras-chave: Arquitetura orientada a serviços. Sistemas embarcados. Redes de comunicação industriais. Semânticas de skills. Marcadores fiduciais.

ABSTRACT

The reference architectural model for Industry 4.0 recommends using the communication standard OPC UA, which establishes component virtual representation techniques based on the abstraction and on the description of their functionalities. One of the use cases is denominated adaptable factories, which are those that spare the need for reconfiguration in their processes when introducing or modifying production components. In this context, this coursework addresses the modelling and implementation of interoperable and autonomous Plug & Produce components as a solution to an auto (re)configurable production system. It reviews the norms and modelling techniques of OPC UA components and implements a demonstration system, characterized by the auto-detection of fiducial markers using camera devices, image processing components and automated composition of their functionalities, which is done by describing them using skill semantics. The experiment shows that such a system is interoperable and possesses autonomy when executed on different servers. However, by conducting a network and operational resources consumption analysis, the system still needs further development to fulfil the requirements of distributed systems. It is concluded that despite the high complexity resulting from the use of a low-level programming language, the approach of using skill semantics is effective and extensible to self-adaptive production systems.

Keywords: Service-oriented architecture. Embedded systems. Industrial communication networks. Skill semantics. Fiducial markers.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE TABELAS

LISTA DE SÍMBOLOS

M-xys	Sistema coordenado *map* fixo
C_i	Câmera i
F_j	Marcador fiducial j
n_C	Número de câmeras no sistema
$\mathbf{o}$	Vetor origem
$\mathbf{R}$	Matriz de rotação
$\mathbf{A}$	Matriz de transformação homogênea

LISTA DE ABREVIATURAS E SIGLAS

AF	Adaptable Factory
API	Application Programming Interface
AAS	Asset Administration Shell
AWS	Amazon Web Services
EAS	Evolvable Assembly Systems
CLP	Controlador Lógico Programável
CPS	Cyber-Physical System
CSV	Comma Separated Values
DHCP	Dynamic Host Configuration Protocol
DNS	Domain Name System
EC2	Elastic Compute Cloud
ERP	Enterprise Resource Planning
GDS	Global Discovery Server
I4.0	Indústria 4.0
IHM	Interface homem-máquina
IIoT	Industrial Internet of Things
IoS	Internet of Services
IoT	Internet of Things
JSON	JavaScript Object Notation
LDS	Local Discovery Server
LDS-ME	Local Discovery Server with Multicast Extension
M2M	Machine to Machine
MEF	Máquina de estados finitos
MES	Manufacturing Execution System
MRP	Manufacturing Resources Planning
MSB	Manufacturing Service Bus
NCS	Networked Control Systems
OPC UA	OPC Unified Architecture

P&P	Plug & Produce
PnP	Plug and Play
PPR	Produto-processo-recurso
RAMI4.0	Reference Architectural Model Industrie 4.0
RPC	Remote Procedure Call
SDK	Software Development Kit
SDN	Software Defined Network
SOA	Service-Oriented Architecture
TCP	Transmission Control Protocol
UML	Unified Modeling Language
VPN	Virtual Private Network
XML	Extensible Markup Language

SUMÁRIO

1 INTRODUÇÃO

A automação industrial foi impulsionada pela introdução dos Controladores Lógicos Programáveis (CLPs), por meio dos avanços na eletrônica no fim dos anos 1960, durante a Terceira Revolução Industrial. O controle da manufatura passou, nessa época, a ser assistido por computadores com o emprego das tecnologias da informação (SACOMANO et al., 2018).

Somado à produção enxuta (lean manufacturing) da época, a informática industrial possibilitou o surgimento de novas técnicas de controle de produção e informatização de processos, como o Manufacturing Resources Planning (MRP), para o controle da necessidade de componentes e gerenciamento de recursos industriais, e como o Enterprise Resource Planning (ERP), que integra todo o processo industrial à empresa (MOURTZIS et al., 2015).

O início do século XXI foi marcado pela forte presença da Internet, que trouxe avanços nas tecnologias de informação e comunicação, concretizando a ocorrência da Quarta Revolução Industrial, quando as empresas foram forçadas a traçarem novas estratégias de negócios. Os conceitos de produtividade, flexibilidade e qualidade foram, então, colocados em novos patamares, os quais podem ser gerenciados remotamente (SACOMANO et al., 2018).

Atualmente, os avanços tecnológicos introduzem sistemas embarcados de plataforma livre no mercado, proporcionam avanços nos sistemas de comunicação e introduzem arquiteturas computacionais aos processos industriais. Como resultado, a interoperabilidade entre componentes de fabricantes distintos é diretamente afetada, vindo a funcionar de forma mais coordenada e integrada, por exemplo, através dos sistemas remotos de supervisão e dos Networked Control Systems (NCS), que utilizam uma rede de comunicação para interconectar os dispositivos da planta de um sistema: sensores, controladores e atuadores (DE SÁ et al., 2018).

A integração de sistemas físicos, através da computação e das redes de comunicação, é denominada Cyber-Physical System (CPS), cujas abstração e arquitetura visam permitir a composição de sistemas heterogêneos na forma de Plug and Play (PnP). Isso permite que a conexão de um componente específico de qualquer fabricante seja automaticamente integrada com os demais componentes existentes no sistema (BAHETI; GILL, 2011).

Para que a integração de sistemas seja possibilitada, é necessário que haja um controle sobre a concepção das arquiteturas dos modelos de informação utilizados pelos fabricantes e desenvolvedores de tecnologia. Com isso, na Alemanha, o Ministério Federal da Economia e da Proteção do Clima (à época, Ministério Federal da Economia

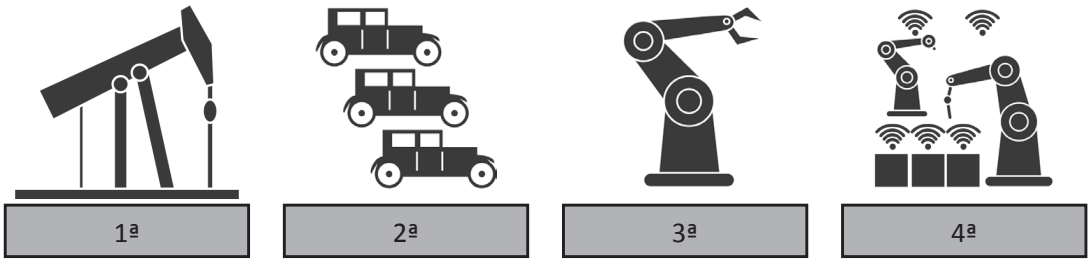
e Energia), decide concretizar a ideia central da Quarta Revolução Industrial, as problemáticas e demandas trazidas pelos sistemas modernos e os requisitos de integração de sistemas em um projeto global denominado *Platform Industrie 4.0* (Plataforma Indústria 4.0), que foi apresentado em 2011 durante a Feira de Hannover.

Tal projeto visa estabelecer uma forma para que sistemas automatizados que controlam os equipamentos industriais se comuniquem com máquinas e humanos, otimizando, dessa forma, todo o processo de produção. Além dos CPSs, a Indústria 4.0 (I4.0) se baseia na interconexão de dispositivos e serviços, de forma que os processos industriais possam se tornar mais descentralizados e mais interoperáveis através de conceitos como Internet of Things (IoT) e Internet of Services (IoS) (SACOMANO et al., 2018). Uma comparação entre os tópicos da I4.0 e as outras três revoluções industriais antecessoras, é mostrada na Figura 1.

A partir da interconexão de dispositivos e serviços, a I4.0 propõe a criação de ecossistemas digitais para substituir as rígidas e fixas cadeias de valores industriais, por redes de valores flexíveis, dinâmicas e conectadas globalmente através da Internet. Os modelos de negócio passam, assim, a serem impulsionados por dados, de forma que a transparência e acesso dos clientes ocupem o primeiro plano da negociação, colocando o produto no centro do palco em comparação com os paradigmas industriais herdados da Terceira Revolução Industrial.

Para garantir o sucesso dos negócios dentro do mercado competitivo, os ecossistemas digitais devem focar em: disponibilidade, transparência e acesso aos dados, além de garantir abertura para demonstrar diversidade, pluralidade e apoio a todos os atores mercantis competitivos. Ainda, coloca-se três campos de ação estratégicos para a garantir a implementação bem-sucedida da I4.0, bem como a forte interconexão: autonomia, interoperabilidade e sustentabilidade (ALEMANHA, 2019).

Figura 1 – As quatro revoluções industriais



Fonte: Sacomano et al. (2018, p. 28).

O processo de digitalização da indústria envolve a comunicação e representação virtual de componentes I4.0, e a Plattform Industrie 4.0 coloca o modelo de referência Reference Architectural Model Industrie 4.0 (RAMI4.0) como proposta de concepção arquitetônica para os componentes industriais, definido por uma forma precisa de descrever os componentes e as redes de produção que os envolvem.

A primeira versão do RAMI4.0 foi publicada pela plataforma através de uma junção das associações alemãs Zentralverband Elektrotechnik- und Elektronikindustrie e.V (ZVEI), Verband Deutscher Machinen- und Anlagenbau (VDMA) e pela Associação Federal Bundesverband Informationswirtschaft, Telekommunikation und neue Medien (BITKOM). Essa versão descreve precisamente a arquitetura de equipamentos de produção conforme os requisitos da I4.0 (DEUTSCHES INSTITUT FÜR NORMUNG, 2016).

No desenvolvimento de sistemas de software complexos utiliza-se o conceito de abordagem orientada por modelos, separando a descrição funcional de um componente de sua implementação, resultando na caracterização dos componentes I4.0 por uma comunicação de arquitetura orientada à definição de serviços, denominada Service-Oriented Architecture (SOA). Os componentes também integram uma representação virtual que integra informações semânticas de suas funcionalidades ao componente físico, descrevendo a relação entre os significados e suas denotações, ou, posto de outra forma, *para o quê servem* (PAUKER et al., 2016).

Segundo Jammes e Smit (2005), SOA representa o "[. . .] conjunto de princípios arquitetônicos para construir não apenas sistemas *autônomos*, mas também *interoperáveis*" (p. 717), de tal forma que tais sistemas sejam um independente do outro e provém funcionalidades auto-contidas, garantindo a *autonomia*. A *interoperabilidade* entre os sistemas é atingida à medida que se abstrai claramente a interface que um serviço expõe no ambiente de implementação. De forma objetiva, SOA estabelece conciliar as propriedades autônomas e interoperáveis dos sistemas, sem que haja contradições entre si.

Além da descrição de suas interfaces quanto às funcionalidades, um serviço pode ter requisitos, propriedades ou capacidades expressadas através de políticas, negociadas em tempo de execução. Para garantir a interoperabilidade entre sistemas, a concepção de um serviço começa em determinar como que se encaixa em contextos genéricos e de maior dimensão em relação ao processo que descreve.

Os detalhes da implementação de um serviço não fazem parte da interface apresentada ao ambiente, sendo a arquitetura SOA centrada nos processos de negócios, ao invés de ser centrada nas tecnologias utilizadas, tal que a implementação das funcionalidades do serviço se mantém totalmente encapsulada e pode ser modificada sem que o usuário seja afetado. Entretanto, ainda existe uma pequena dependência entre serviços no processo de comunicação, baseado na troca de

documentos e, portanto, cada serviço deve conhecer o padrão de modelagem de informação contido nesses documentos, padrão que deve ser comum a todos os outros serviços (JAMMES; SMIT, 2005).

Tendo em vista os requisitos da arquitetura SOA, o modelo de referência RAMI4.0 propõe o uso do padrão de comunicação OPC Unified Architecture (OPC UA), em todas as camadas da pirâmide de automação, por ser um padrão desenvolvido para prover interoperabilidade entre os processos através da abstração de serviços. Por se tratar de um padrão, o OPC UA não especifica uma Application Programming Interface (API) para troca de informações, estabelecendo apenas o formato das mensagens, de forma que não haja dependência entre o padrão e o protocolo de comunicação utilizados.

A codificação e decodificação de mensagens é feita por meio do mapeamento aplicado nas pilhas de comunicação envolvidas no processo, o quê permite o uso de outras tecnologias para o transporte de dados, desde que se implemente o mapeamento definido pelo padrão, como especificado em OPC Foundation (2021c).

A flexibilidade provida pela arquitetura utilizada no padrão OPC UA permite o uso de servidores agregados por meio de encadeamento. Um servidor agregador é aquele que une em si um ou mais servidores OPC UA e disponibiliza as informações, ou parte das informações dos servidores agregados, em seu espaço de endereçamento, sendo, então, o suficiente que um cliente acesse o servidor agregador para ter acesso aos dados de outros componentes. Geralmente, utiliza-se a camada de Manufacturing Execution System (MES) como um servidor agregador (LEITNER; MAHNKE, 2006).

Todas as interações entre servidores e clientes ocorrem por meio dos serviços definidos pelo OPC UA, os quais são classificados em coleções de chamadas de procedimentos remotos, Remote Procedure Call (RPC), implementadas pelos servidores. Como exemplo de serviços OPC UA, tem-se o conjunto de serviços de descobrimento (Discovery Service Set), que permite o cliente procurar e descobrir os *endpoints* de um servidor e também ler a configuração de segurança de cada um desses *endpoints*. O conjunto de serviços de atributos (Attribute Service Set) define para os clientes serviços de escrita e leitura de atributos de nós de servidores, como uma variável, a qual é modelada como atributo (OPC FOUNDATION, 2021b).

A Plattform Industrie 4.0 separa a implementação da I4.0 em diferentes casos de uso, os quais auxiliam na visualização e indicação de decisões acerca dos investimentos nas tecnologias relacionadas, o quê se torna possível pela caracterização dos casos de uso em nove diferentes cenários de aplicação, que determinam uma descrição genérica e geral de um problema ou de um desafio no contexto da produção. Dentre eles, Adaptable Factory (AF) é um cenário de aplicação de fábricas adaptáveis, estabelecendo como que um recurso específico de produção pode ser readaptável a diferentes processos industriais ou, em outras palavras, AF esboça uma linha de

produção flexível que pode ser reconfigurada durante tempo de execução (ALEMANHA, 2016).

O cenário de aplicação AF é ideal quando fabricantes necessitam reajustar as linhas de produção para uma nova variante de produto ou quando desejam implementar novos recursos de produção, tornando o trabalho significativamente menos manual e, consequentemente, reduzindo drasticamente os custos de implementação. O cenário pode, ainda, ser dividido em diferentes casos de uso, como o caso da integração de um componente em uma unidade de produção por meio do uso de Plug & Produce (P&P), através da orientação a serviços (ALEMANHA, 2017).

O termo Plug & Produce (P&P) tem origem na tecnologia Plug and Play (PnP), utilizada nos computadores, que define um padrão de arquitetura de serviços de dispositivos periféricos como teclado, mouse, câmera e impressora. Dessa forma, a troca de um periférico por outro, de mesma natureza e de diferentes modelos ou fabricantes, não implica na reinicialização ou reconfiguração manual do sistema. Esse mesmo conceito é integrado às aplicações industriais por meio de P&P, em que a conexão de novos componentes resulte em mínimo ou nenhum trabalho manual, melhorando a flexibilidade de produção (ALEMANHA, 2017).

A sinergia dos conceitos abordados até aqui formam a base para o desenvolvimento de fábricas adaptáveis, as quais consistem de unidades de produção modulares, inteligentes e interoperáveis, podendo ser facilmente recompostas ou estendidas para suprir as demandas de consumidores. Neste cenário, por meio da integração de P&P em uma rede de comunicação industrial OPC UA, modela-se e implementa-se módulos de produção com suas propriedades e capacidades semanticamente bem definidas, definindo o tema deste trabalho.

1.1 OBJETIVO GERAL

Através do levantamento de técnicas e métodos de modelagem de componentes P&P, com base nas documentações e relatórios técnicos, descrever a aplicação dos conceitos principais em uma rede industrial OPC UA. Seguidamente, esquematizar uma aplicação demonstrativa composta por dispositivos de câmeras e por componentes de software de processamento de imagem para, finalmente, executar a implementação, utilizando bibliotecas de código aberto, em dispositivos embarcados.

1.2 OBJETIVOS ESPECÍFICOS

- Introduzir os princípios básicos de modelagem de informação no padrão de comunicação OPC UA.
- Estudar as normas e técnicas para modelagem de componentes com capacidade P&P.

- Conhecer as ferramentas para implementação no padrão OPC UA dos componentes modelados.
- Esquematizar o processo de descobrimento e interconexão de dispositivos OPC UA.
- Aplicar a biblioteca de código livre open62541 para implementar uma rede de componentes com a funcionalidade de detecção da posição de marcadores fiduciais, composta por câmeras e um processador de imagens.
- Demonstrar a funcionalidade de P&P tanto por meio do sistema conectado por uma rede local, quanto por uma rede virtual distribuída.
- Avaliar as características positivas e negativas da metodologia proposta com base em um conjunto de requisitos.

2 FUNDAMENTAÇÃO TEÓRICA

O padrão de comunicação industrial OPC UA é especificado pela fundação OPC Foundation, que descreve minuciosamente as suas características da modelagem de informação, conexão, segurança, entre outras. Para a execução do trabalho proposto, é de extrema importância o entendimento do espaço de endereçamento em OPC UA, abordado na Seção 2.1, que introduz os conceitos fundamentais sobre o funcionamento do padrão e dos recursos de interesse. Os Apêndices A, B e C trazem informações adicionais, não menos importantes, a respeito das características de modelagem e representação gráfica do padrão.

Os conceitos acerca do tema P&P vêm sendo documentados pela organização Plattform Industrie 4.0, que especifica métodos e regras de modelagem de sistemas que o implementam. Assim, a Seção 2.2 traz uma visão geral da ideia central do conceito, algumas abordagens de implementação e, principalmente, o desenvolvimento de trabalhos já executados pela comunidade acadêmica, bem como o seu uso em conjunto com o padrão OPC UA.

Por fim, a Seção 2.3 introduz, resumidamente, os marcadores fiduciais, sua funcionalidade e o método de detecção utilizando câmeras, que fundamenta o exemplo de aplicação proposto neste trabalho.

2.1 OPC UNIFIED ARCHITECTURE

Voltado à troca de informação entre dispositivos, monitoramento e controle de processos industriais, o desenvolvimento do padrão OPC UA tem como base a aplicação em componentes de toda natureza, por exemplo, software, sensores, atuadores e sistemas de controle, sendo também utilizado em interfaces como MES, ERP, Industrial Internet of Things (IIoT), Machine to Machine (M2M) e I4.0. De forma geral, a sua infraestrutura pode ser dividida entre modelos de informação, de mensagem, de comunicação e de conformidade, garantindo a interoperabilidade entre sistemas (OPC FOUNDATION, 2017a).

A comunicação entre sistemas distribuídos e a modelagem de dados, orientada a objetos e com sistemas de tipos extensíveis, caracterizam os principais requisitos de projeto para o OPC UA, provendo escalabilidade de modelos simples e complexos. No processo da comunicação, mensagens de requisições e respostas são trocadas por meio do paradigma cliente-servidor e, ainda, é também possível utilizar o padrão publish-subscribe, tal que, como consequência da abstração do modelo de comunicação, não há dependência de um protocolo de transporte específico, tornando-a flexível (MAHNKE; LEITNER, 2009).

A descrição de processos e recursos no padrão OPC UA é implementada por meio dos chamados modelos de informação, caracterizados por especificações, definidas pela OPC Foundation e denominadas UA Specifications. A fundação utiliza um conjunto base de especificações sob as quais são definidas as outras, como mostra a Figura 2 uma hierarquia simplificada de blocos de especificações, em que se descreve as seguintes:

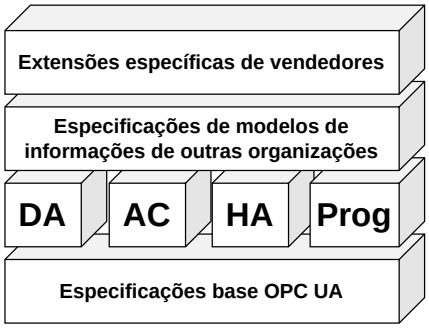
Data Access (DA) define extensões específicas para acesso a dados de processos.

Alarms & Conditions (AC) especifica um modelo avançado para gerenciamento de processos de alarme e monitoramento de condições.

Historical Access (HA) define um mecanismo para acessas dados e eventos históricos arquivados.

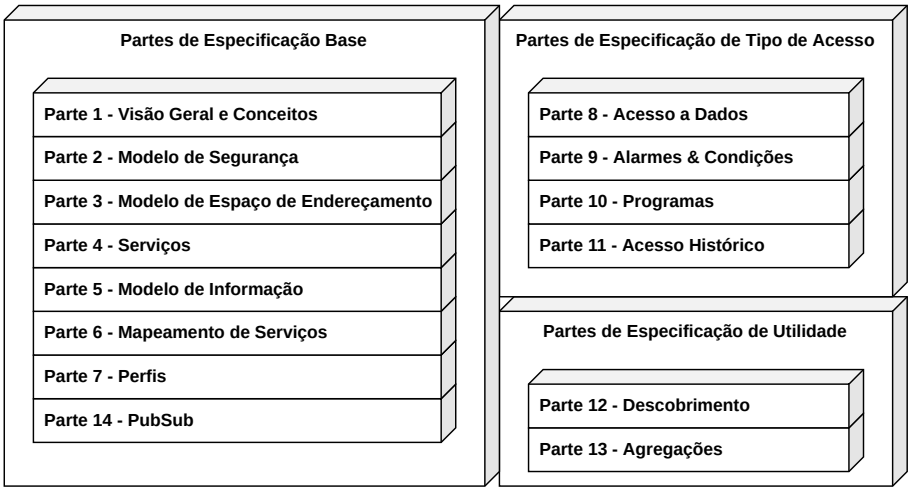
Programs (Prog) especifica um mecanismo para iniciar, manipular e monitorar a execução de programas.

Figura 2 – Hierarquia simplificada de especificações OPC UA



Fonte: Mahnke e Leitner (2009, p. 11).

Figura 3 – Grupos de especificações OPC UA



Fonte: OPC Foundation (2017a, p. 7).

De forma geral, as especificações OPC UA são compostas por diferentes partes e grupos, mostrados na Figura 3, sendo todos documentados[1] pela OPC Foundation.

[1] https://reference.opcfoundation.org/

- As Partes de 1 a 7 e a Parte 14 especificam os recursos de núcleo do OPC UA, que definem a estrutura do espaço de endereçamento e os serviços operantes.
- A Parte 14 define o padrão publish-subscribe, além de cliente-servidor definido pelos Serviços na Parte 4.
- As Partes de 8 a 11 definem as especificações já previamente descritas e representadas na Figura 2.
- A Parte 12 define os mecanismos de descobrimento entre servidores OPC UA.
- A Parte 13 descreve os meios de agregação de dados.

Para compreender o desenvolvimento e implementação de uma aplicação em um componente OPC UA, uma visão geral da arquitetura, do espaço de endereçamento, do modelo de informação e dos serviços será abordada ao longo dessa Seção. O padrão apresenta, entretanto, uma vasta documentação com rica estrutura de modelagem, porém, para fins de trabalho de graduação, não é necessário abordar e implementar todos os recursos, mas apenas aqueles vitais para estruturação da aplicação proposta.

2.1.1 Camadas de software

Uma aplicação OPC UA típica é composta de três camadas de software, de forma que o padrão não seja limitado a apenas uma ou duas linguagens de programação ou até a alguns protocolos de transporte, conforme mostrado na Figura 4. A aplicação é caracterizada por um sistema que expõe ou consome dados por meio do padrão OPC UA, tal que suas funcionalidades são mapeadas por outras duas camadas: pela pilha de dados OPC UA Stack e pela biblioteca de desenvolvimento OPC UA Software Development Kit (SDK) (MAHNKE; LEITNER, 2009).

A pilha de dados UA Stack é responsável apenas pela implementação dos canais de comunicação, o quê é feito por meio de mapeamentos de protocolos de transporte, conforme definido em OPC Foundation (2021c). Essa camada é, ainda, subdividida em um conjunto de implementações para ligação das mensagens com a API da pilha UA Stack, as quais são processadas por blocos de codificação e decodificação, compostos pela serialização da mensagem em fromato Extensible Markup Language (XML) ou binário, pela camada de segurança e pela camada de transporte, conforme mostra a Figura 4.

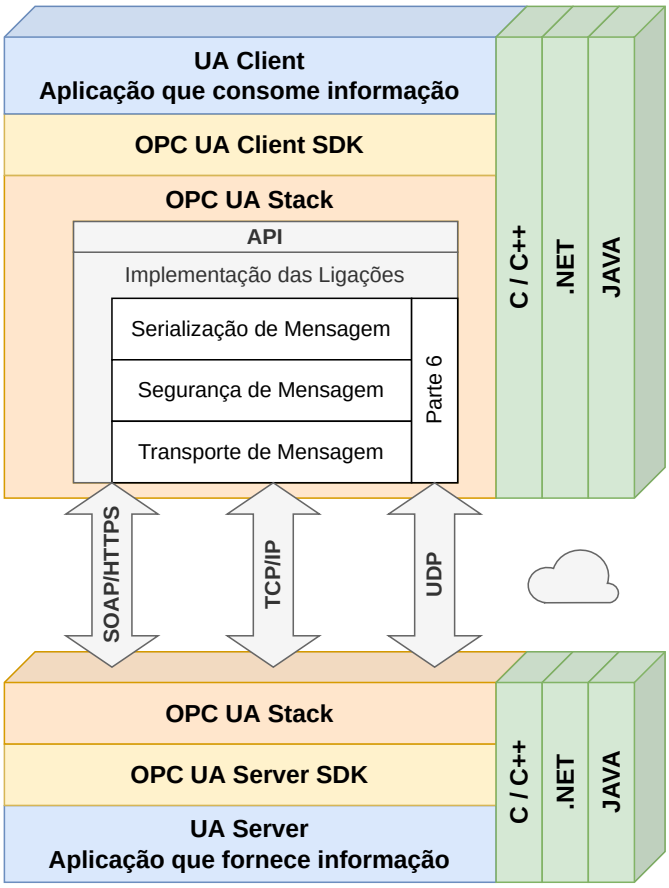
Em seguida, introduz-se como que as mensagens são estruturadas no padrão OPC UA, ou seja, como é definido o espaço de endereçamento dos servidores UA Server e como o cliente pode acessá-los. Deve ser notado que o entendimento do espaço de endereçamento é vital para o uso de um OPC UA SDK, pois as bibliotecas utilizam a mesmas estruturas e nomenclaturas.

2.1.2 Espaço de endereçamento

O padrão OPC UA introduz uma notação gráfica própria para o espaço de endereçamento, baseada no padrão Unified Modeling Language (UML), que, basicamente, identifica as classes de nós e os tipos de referências entre elas, definidas em OPC Foundation (2022b). No Apêndice A encontra-se as principais notações que serão utilizadas ao longo dessa Seção, em que as classes de nós NodeClass ainda serão abordadas.

Em geral, a modelagem de informação em OPC UA, sempre feita no lado do servidor, tem como princípio a utilização de técnicas de orientação a objetos, incluindo hierarquias de tipo e herança, tal que o tipo da informação é exposto e, logo, pode ser acessado da mesma forma que as instâncias, provendo extensibilidade às aplicações. Outro princípio é o uso de malhas de redes de nós, que permitem conectar a informação em várias direções, não havendo limitações em como a modelagem deve ser propriamente feita (MAHNKE; LEITNER, 2009).

Figura 4 – Camadas de software em OPC UA

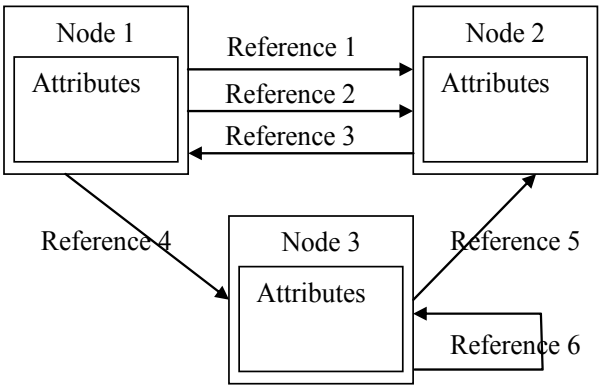


Fonte: adaptado de Mahnke e Leitner (2009, p. 14 e 15).

2.1.2.1 Nós e referências

O conceito base da modelagem de informação em OPC UA é a referência entre nós, descritos por atributos, que podem ser de qualquer natureza, ou seja, de diferentes classes NodeClass, sendo instâncias ou tipos. Os atributos de um nó dependem da sua classe NodeClass, entretanto, os atributos de 1 a 7, listados no Apêndice B, são comuns a todos os nós. A Figura 5 mostra a notação de uma relação básica entre diferentes nós.

Figura 5 – Notação de nós, atributos e referências



Fonte: Mahnke e Leitner (2009, p. 22).

Um nó é identificado no servidor por um atributo NodeId unívoco, base do endereçamento de informação, o qual é definido por um identificador textual ou numérico e por um Namespace, tal que um nó pode conter NodeIds alternativos, porém apenas um canônico. O NodeId é especialmente utilizado no serviço de busca de nós em um servidor, entretanto, também há o atributo de busca BrowseName, uma estrutura que define um texto não localizado e um Namespace para referenciar o nó.

As referências descrevem a relação entre dois nós, unicamente descritas pelo nó de origem, pelo nó de destino, pela semântica, ou seja, pelo tipo de referência, e pela direção da referência, simétrica, assimétrica ou hierárquica. Na prática, pensa-se em uma referência como um ponteiro, cujo objeto da ponta é o NodeId do nó referenciado, que pode estar no mesmo ou em outro servidor. A notação gráfica e os tipos de referências são apresentados no Apêndice A. Os tipos base de referências, dados e eventos em OPC UA são representados no Apêndice C.

2.1.2.2 Objetos, variáveis e métodos

Assim como na programação orientada, objetos são conjuntos de variáveis e métodos, ainda que, em OPC UA, podem disparar eventos, notificando clientes subscritos. Objetos são nós que estruturam o espaço de endereçamento, contendo apenas dados descrevendo o nó com atributos, como DisplayName e Description, tal que seus valores são expostos através de variáveis e, suas funcionalidades, por meio

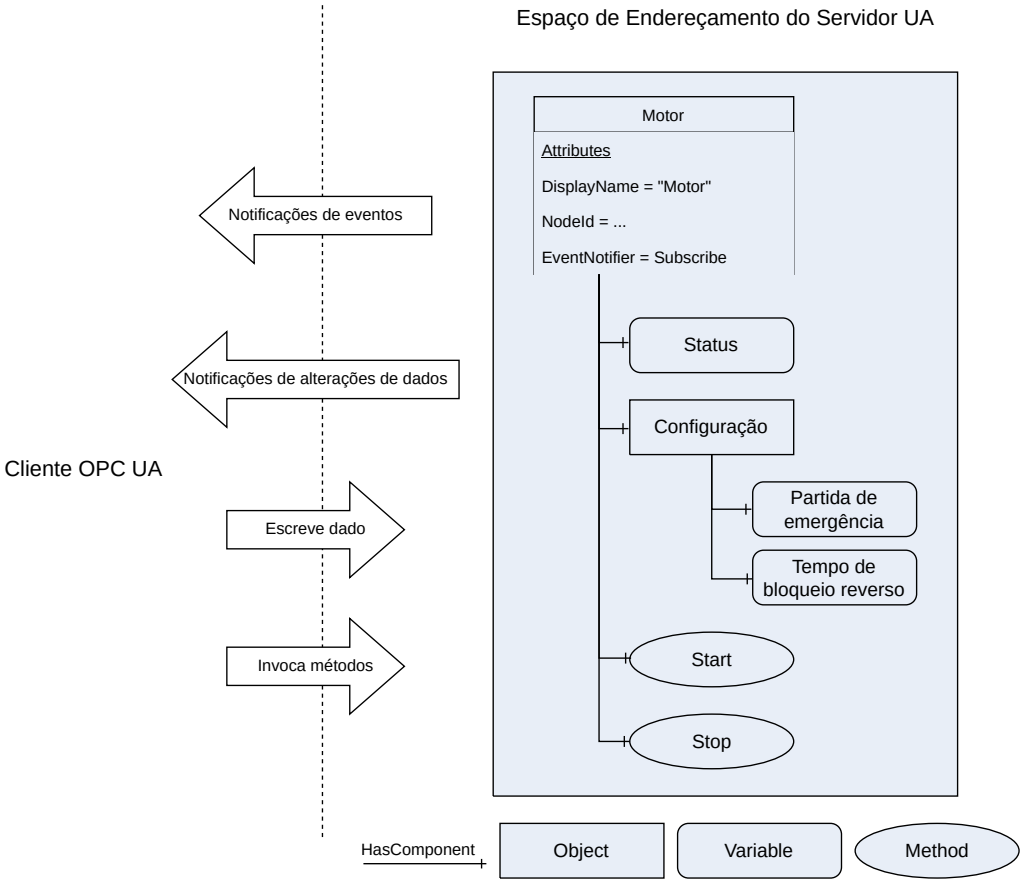
de métodos.

As variáveis representam valores de um tipo específico, como o valor de temperatura de um sensor, de forma que os clientes podem ler ou escrever o valor e, até, se subscreverem para serem notificados quando o valor for alterado. Uma variável pode, também, ser usada para expor qualquer informação que não é exposta pelas referências ou atributos de um nó, como dados de configuração ou informações adicionais descrevendo o nó.

Métodos podem ser chamados por clientes, especificando os argumentos de entrada e saída, ainda, idealmente sua execução deve ocorrer relativamente rápida, logo, para processos de longa duração, utiliza-se o modelo de informação de programas, especificado em OPC Foundation (2022c). Para serem invocados, os clientes devem utilizar o serviço de chamada Call Service, cuja resposta contém o dado de saída, conforme especificado em OPC Foundation (2021b).

A Figura 6 mostra a modelagem simplificada de um motor no espaço de endereçamento de um servidor e as possíveis interações com o espaço do cliente, nota-se que os parâmetros de configuração do motor são agrupados em um outro objeto.

Figura 6 – Visão geral de objetos, variáveis e métodos



Fonte: Mahnke e Leitner (2009, p. 31).

Method é um tipo de dado que realiza a entrada e saída de chamada por

meio de duas variáveis padrões (propriedades), respectivamente, InputArguments e OutputArguments, ambas do tipo de dado `Argument[]`, ou seja, vetores de argumentos, ordenados de acordo com a ordem de passagem, cuja estrutura é definida pelas variáveis Name, DataType, ValueRank, ArrayDimensions e Description. Os métodos e variáveis sempre pertencem a um objeto e são sempre invocados no contexto desse objeto, sendo o método referenciado por meio de HasComponent e, a variável, podendo ser referenciada tanto por meio de HasComponent quanto HasProperty.

Em OPC UA há, ainda, dois tipos de variáveis: variáveis de dados e propriedades, tal que ambos representam dados, porém a variável de dado é utilizada para dados de objetos, como a temperatura de um sensor ou o fluxo de um transmissor, podendo, essas, serem estendidas com outras sub-variáveis ou com propriedades descritivas. As propriedades são utilizadas para representar características de um nó, como a unidade de medida da temperatura de um sensor, ou seja, são simples e não extensíveis.

Em suma, variáveis de dados devem ser referenciadas, por meio de HasComponent, por um objeto, por um tipo de objeto, por uma variável ou por um tipo de variável, nesse último caso, constituindo a possibilidade de expor dados complexos. As variáveis de dados estão, ainda, sempre relacionadas, direta ou indiretamente, com o tipo base BaseDataVariableType.

2.1.2.3 Serviços

A interface de comunicação no nível de aplicação em OPC UA é definida por meio de serviços, em OPC Foundation (2021b), que são métodos utilizados por clientes para acessar os dados de um modelo de informação de um servidor, cujo funcionamento é caracterizado pelo padrão de mensagens de requisições e respostas, logo, as invocações de serviços são assíncronas por definição. A abstração da definição dos serviços pode ser aplicada a diferentes mecanismos de transporte, atendendo ao requisito de independência do protocolo e da linguagem de programação, em que a implementação das APIs da camada de serviços é feita nas pilhas de comunicação OPC UA Stacks, como representado na Figura 4 (MAHNKE; LEITNER, 2009).

Os serviços são organizados em conjuntos de acordo com a sua funcionalidade, tal que, de acordo com o seu perfil, um servidor estabelece se suporta, ou não, um determinado conjunto de serviço, perfis os quais são definidos em OPC Foundation (2017b). Cada serviço possui, ainda, um cabeçalho de requisição e resposta para suas mensagens, como `authenticationToken`, `timestamp` e `requestHandle`, que são parâmetros comuns de requisição para todos os serviços.

Uma vez que uma conexão é estabelecida entre o cliente e um servidor, utiliza-se o conjunto de serviços View Service Set para navegar pelo espaço de endereçamento do servidor, ou por um subespaço, denominado View, por meio dos

serviços de busca, como `Browse` ou `BrowseNext`, que retornam as referências de um nó específico. Para acessar os atributos de um nó, utiliza-se o conjunto Attribute Service Set, que dispõe de serviços de leitura e escrita, como especificado em OPC Foundation (2021b).

Os resultados da chamada de um serviços são retornados em dois níveis de resposta, sendo uma que indica o status da chamada do serviço, parâmetro `serviceResult`, e outra que indica o status de cada operação requisitada pelo mesmo serviço, através de parâmetros específicos, tal que ambas são definidas por códigos de status, separados em códigos de sucesso e códigos de falha. Como exemplo de aplicação, um serviço retorna o código `Bad_CertificateUntrusted` se as credenciais de usuário de um cliente estão comprometidas, impossibilitando criar um canal seguro de comunicação (OPC FOUNDATION, 2021b).

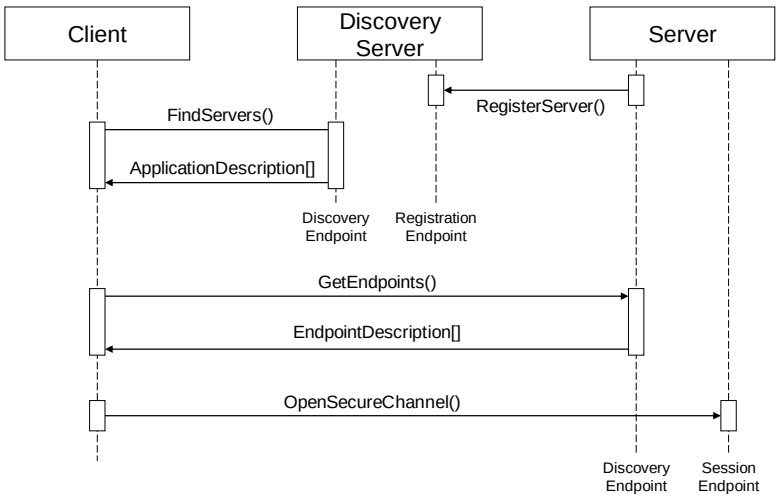
Um *endpoint* é definido em OPC Foundation (2021b) como "[. . .] um endereço físico disponível em uma rede, que permite clientes a acessarem um ou mais serviços providos por um servidor", tal que "Cada servidor pode ter múltiplos endpoints"e "Cada endpoint inclui um HostName" (p. 2). Essa definição introduz o conjunto de serviços de descobrimento, Discovery Service Set, utilizado para descobrir os endpoints implementados por um servidor, permitindo, também, verificar as configurações de segurança para cada endpoint.

Os serviços de descobrimento podem ser implementados em servidores individuais ou em um servidor dedicado, Discovery Server, global ou local, como especificado em OPC Foundation (2018a). Entretanto, é necessário que cada servidor tenha um endpoint de descobrimento, DiscoveryEndpoint, que permite clientes a acessarem os serviços de descobrimento sem a necessidade de iniciar uma sessão de comunicação e sem segurança de mensagem.

Quando utilizando um servidor de descobrimento dedicado, os servidores devem se registrarem nesse servidor, já conhecido, por meio do serviço `RegisterServer`, para que os clientes possam, então, obter uma lista de servidores registrados na rede, por meio do serviço `FindServers`. Esse processo é ilustrado no diagrama de sequência da Figura 7, em que a chamada do serviço `OpenSecureChannel` define, de fato, a conexão do cliente com o endpoint requisitado em um servidor.

Após o processo de descobrimento, o cliente pode salvar os dados dos endpoints e utilizá-los para futuras conexões, poupando-o tempo. Entretanto, as configurações do servidor podem mudar e, se ocorrer discordâncias com as informações guardadas pelo cliente, deve-se realizar o processo de descobrimento novamente.

Figura 7 – Processo de descobrimento



Fonte: OPC Foundation (2021b, p. 10).

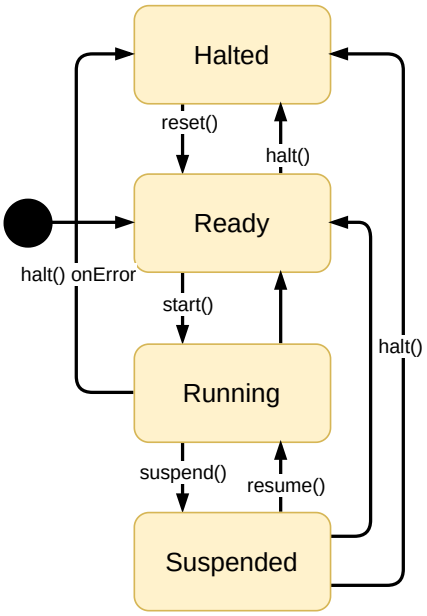
2.1.2.4 Programas

Os processos industriais são marcados pela execução de diferentes funções, constituídas de operações sistemáticas nas linhas de produção, que podem ser invocadas utilizando sistemas de controle supervisórios ou por interface homem-máquina (IHM). Em OPC UA, essas funções são definidas por meio de métodos e programas, porém, que possuem diferentes escopos, comportamentos, tempos de vida e complexidades, pois, como já abordado, métodos são processos que são pensados para rápidas execuções.

Programas complementam a baixa complexidade dos métodos, permitindo o uso de funcionalidades de estados, como a operação de uma máquina de ferramentaria, que também podem ser invocados e gerenciados por clientes. Entretanto, os programas são acionados por meio de métodos de transições, pois são representados por máquinas de estados finitos (MEFs), provendo um conjunto padrão de quatro estados base, expansíveis, representados na Figura 8 com suas respectivas possíveis transições.

Os resultados de um programa podem ser divididos em intermediários e finais, tal que o primeiro está associado aos dados são gerados em as transições de estados não terminais, já os resultados finais são aqueles retornados quando o programa termina sua execução. Cada transição de estado de um programa pode ser associada a um item de dado de resultado diferente ou, ainda, compartilhar o item com um outro conjunto de estados. Os dados são entregues ao cliente por meio dos eventos gerados, logo, é necessário que o cliente monitore tais eventos (OPC FOUNDATION, 2021a).

Figura 8 – Estados e transições de programa



Fonte: adaptado de OPC Foundation (2021a, p. 5).

2.1.3 open62541

O padrão OPC UA é, também, denominado pela Comissão Eletrotécnica Internacional de norma IEC 62541, o quê dá o nome de open6241 à biblioteca de código aberto, licenciada sob Mozilla Public License v2.0, que implementa o padrão utilizando os subconjuntos comuns das linguagens de programação C99 e C++98, tornando-a independente da plataforma de implementação. A biblioteca implementa o protocolo de fila OPC UA binário, bem como o SDK para o cliente e servidor, com suporte ao perfil de servidor de dispositivo micro-embarcado (OPEN62541, 2021).

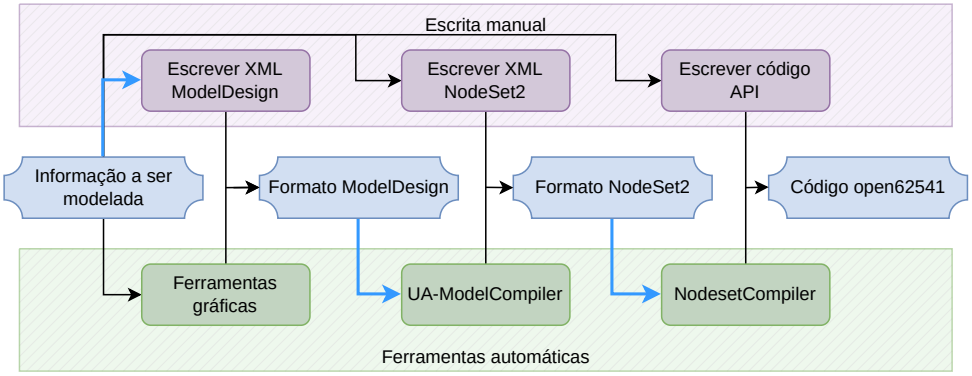
Fundamental para a modelagem de informação, a biblioteca open62541 dispõe do suporte a todos os tipos de nós OPC UA, bem como suporte à herança e instanciação de objetos e variáveis e monitoramento de itens por meio de subscrição a eventos. Outra funcionalidade de destaque é a geração de código a partir de definições padrões em XML, como definidas pela norma IEC 62541, resultando em códigos para tipos de dados complexos e *nodesets*, modelos de informações de servidor.

Uma aplicação em OPC UA é caracterizada por um conjunto de nós, NodeSet, com determinadas relações e funcionalidades que podem ser herdadas de outro conjunto já criado. Isso é feito utilizando a esquematização de aplicações em OPC UA NodeSet XML, cujo objeto de saída é denominado `NodeSet2.xml`, o qual é depois inicializado pela implementação OPC UA.

Profanter (2019) e Profanter (2020) mostram como uma aplicação pode ser desenvolvida utilizando a biblioteca open62541 em conjunto com a modelagem em XML, a qual é interpretada por um compilador de modelo, desenvolvido e mantido pela

própria OPC Foundation. A Figura 9 ilustra o processo de modelagem de informação de NodeSets em OPC UA, conforme proposto pelo autor, onde as linhas destacadas em azul indicam o caminho recomendado.

Figura 9 – Modelagem de informação de NodeSet em OPC UA



Fonte: adaptado de Profanter (2021, p. 85).

A modelagem pode ser feita escrevendo, manualmente, arquivos XML, porém, não é prático para gerar o formato NodeSet2 e nem para gerar o código da API, pois alterações requereriam retrabalho. Logo, utilizam-se as ferramentas UA-ModelCompiler[2] e NodesetCompiler[3] para essas etapas.

2.2 PLUG & PRODUCE

A configuração automática de equipamentos de produção é denominada Plug & Produce (P&P), que é, principalmente, aplicada à integração horizontal dos níveis da piramide de automação, cuja integração com processos de tempo crítico em redes Industrial Ethernet é abordada em Reinhart et al. (2010). Com base nas especificações de Industrial Ethernet, é proposto um modelo de cinco passos para a implementação da funcionalidade de P&P em componentes industriais, os quais ocorrem durante o processo de integração:

1. Conexão física: o dispositivo deve ser conectado à rede, preparada para recebê-lo.
2. Descobrimento: o servidor de gerenciamento detecta a presença do novo dispositivo.
3. Comunicação básica: sem a necessidade de tempo real, o servidor recebe as informações básicas do componente, como a descrição de dispositivo.
4. Avaliação de capacidade: o servidor avalia a identidade, as funcionalidades e requisitos do novo dispositivo com base nas informações providas no passo anterior, o quê envolve combinar os requisitos de produção às capacidades do componente para determinar como (re-)configurar o dispositivo e todo o sistema de produção.

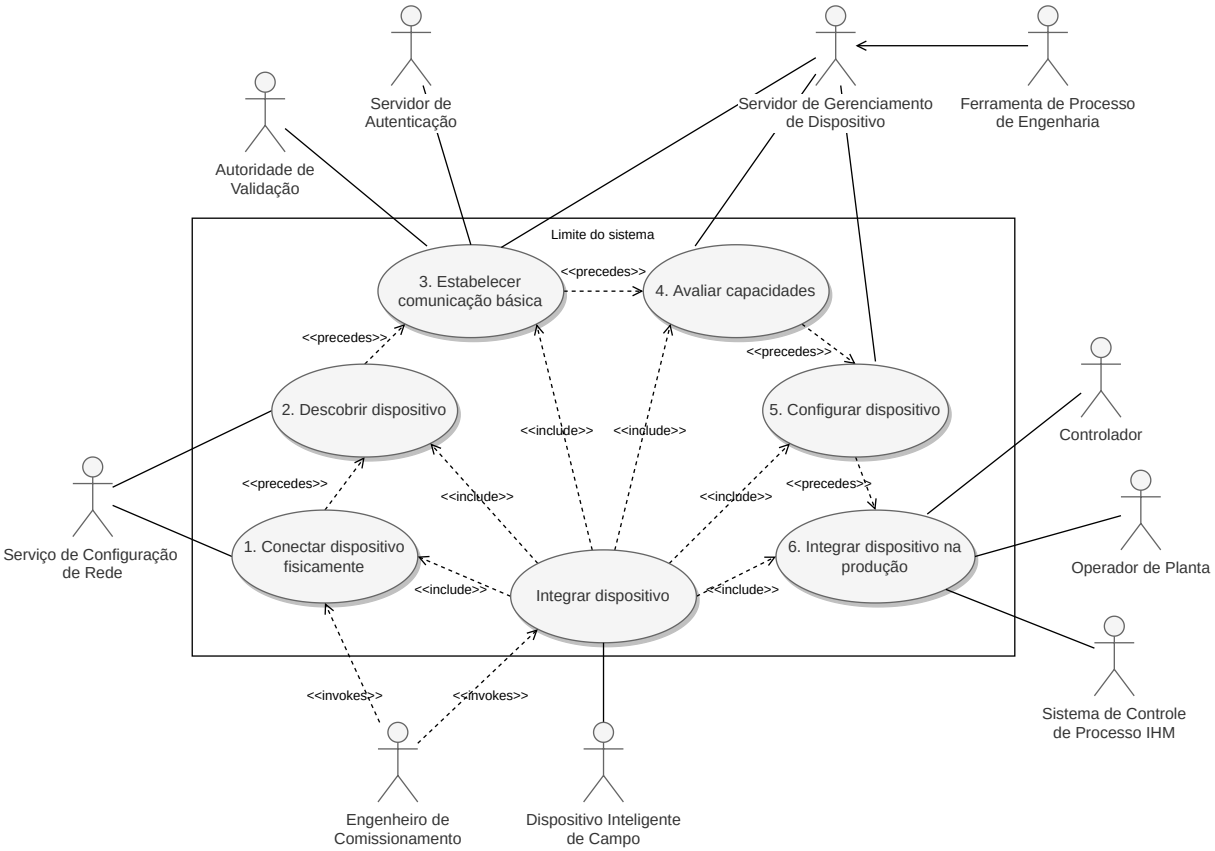
[2] https://github.com/OPCFoundation/UA-ModelCompiler
[3] https://open62541.org/doc/current/nodeset_compiler.html

5. Configuração: a informação de dispositivo é integrada à configuração de rede para possibilitar a comunicação em tempo real, tal que, após, a operação da rede pode ser resumida.

O mesmo modelo é abordado em Alemanha (2017), com a adição de um novo passo, a *integração*, que define a necessidade de reconfiguração dos sistemas MES e ERP, informando os operadores de planta. No contexto, utiliza-se, especialmente, o caso de uso de P&P para dispositivos individuais de campo para operações básicas, cujo cenário de aplicação é o de AF, com o objetivo de reduzir o tempo de comissionamento de dispositivos de campo.

A Figura 10 descreve o modelo de implementação de P&P em um diagrama de casos de uso, destacando os atores envolvidos no sistema. O Serviço de Configuração de Rede provê endereços de IP e máscaras de rede para todos os componentes, realizando, também, o processo de descobrimento, enquanto o Servidor de Gerenciamento de Dispositivos é responsável pela detecção de novos dispositivos por meio da conexão básica, avaliação de capacidades, configuração e por colocá-los em modo operacional.

Figura 10 – Diagrama de casos de uso de P&P para dispositivos de campo



Fonte: Alemanha (2017, p. 12).

O restante da Seção aborda as operações gerais do modelo proposto, alguns dos principais atores envolvidos, com base no estudo de implementação de

componentes P&P em uma rede industrial OPC UA. Para isso, deve-se, primeiro, analisar os processos de descobrimento, modelagem e comunicação com os componentes envolvidos para, finalmente, efetuar a configuração do sistema e integração do dispositivo na produção.

2.2.1 Descobrimento em OPC UA

O padrão OPC UA especifica serviços de descobrimento, apresentados em na Seção 2.1.2.3, os quais podem existir em clientes e servidores em uma mesma máquina ou em diferentes máquinas em diferentes redes. Entretanto, o processo de descobrimento e os mecanismos para clientes e servidores se tornarem descobríveis são especificados em OPC Foundation (2018a), o quê pode ocorrer por meio de três tipos de servidores dedicados de descobrimento:

Local Discovery Server (LDS) uma instância OPC UA que mantém informação de todos os servidores OPC UA disponíveis na mesma máquina em que está sendo executada. Por padrão, o servidor de descobrimento ouve na porta 4840.

Local Discovery Server with Multicast Extension (LDS-ME) um servidor que mantém informação de descobrimento de todos os outros servidores OPC UA em uma mesma sub-rede local com multicast. Esse processo evita a necessidade de especificar previamente endereços de IP para cada máquina.

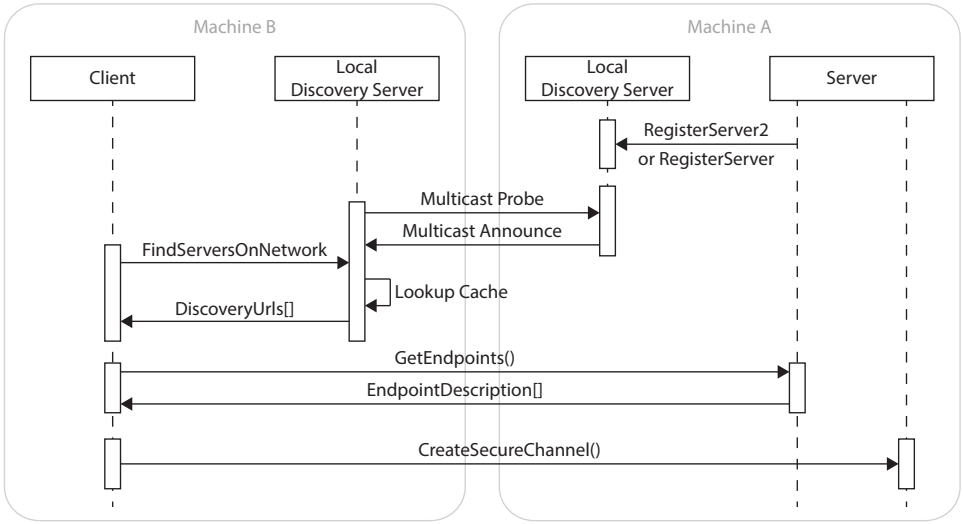
Global Discovery Server (GDS) um servidor para cobertura de grandes sistemas com diversas aplicações, que mantém informação de descobrimento de todos os servidores OPC UA no mesmo domínio administrativo.

O descobrimento por meio de LDS é representado na Figura 7, tal que, utilizando o mesmo conceito, porém com extensão de multicast para a sub-rede local, o processo pode ser estendido para a comunicação entre duas máquinas, contato que cada uma hospede um próprio servidor LDS-ME, conforme mostrado na Figura 11, em que a Máquina B consulta um servidor pré-definido, ou já descoberto, encontrando todos os servidores conhecidos na rede para, finalmente, conectar-se com o endpoint da Máquina A.

Profanter et al. (2017) propuseram uma arquitetura para implementar o servidor LDS-ME utilizando um barramento Manufacturing Service Bus (MSB), responsável por detectar os componentes inteligentes na rede e configurá-los adequadamente, atuando como uma comunicação centralizada. A Figura 12 mostra a arquitetura proposta para a avaliação do descobrimento em OPC UA, tal que, logo depois de conectado, o dispositivo é registrado pela camada de rede 2 com a sua estação de trabalho, recebendo configurações adicionais da estação ou de MSB, o qual é notificado do registro através de multicast.

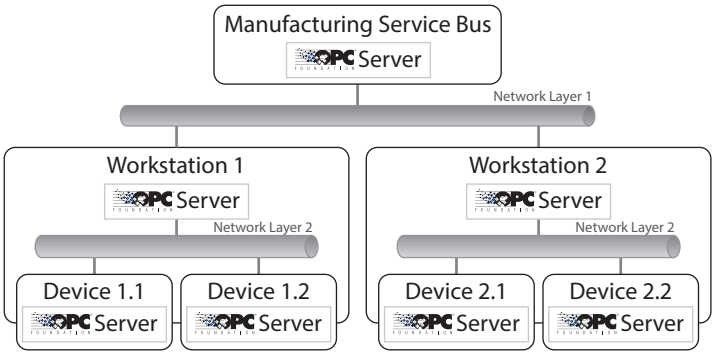
Os autores realizaram a implementação da funcionalidade de LDS-ME na

Figura 11 – Processo de descobrimento com extensão multicast



Fonte: Profanter et al. (2017, p. 5).

Figura 12 – Proposta de arquitetura para P&P em OPC UA



Fonte: Profanter et al. (2017, p. 2).

biblioteca open62541, a qual, à epoca, contava apenas com a funcionalidade de LDS comum, cujo resultado foi a adição de dois serviços: `RegisterServer2`, que permite um servidor registrar seus endereços de descobrimento e capacidades com um servidor de descobrimento, e de `FindServersOnNetwork`, que retorna os servidores detectados por mensagens de multicast. O processo de descobrimento é então detalhado da seguinte forma:

1. Quando um componente é conectado à rede, recebe um endereço de IP em uma sub-rede correspondente por meio de Dynamic Host Configuration Protocol (DHCP), tal que, o servidor OPC UA do componente realiza uma prova de multicast quando a rede é inicializada, recebendo a resposta de todos os outros servidores LDS-ME na mesma sub-rede, com informações de como o servidor LDS-ME pode ser contatado e, finalmente, o componente possa se registrar no servidor. Caso o servidor LDS-ME não esteja disponível, o componente envia

periodicamente uma prova de multicast até encontrá-lo.
2. O novo servidor é guardado em uma lista e, então, instancia-se um cliente OPC UA para se conectar com o novo componente registrado.
3. O cliente realiza o processo de configuração no componente, atualizando os parâmetros de conexão.
4. Se o componente é desligado adequadamente, ele deve desfazer o seu registro no servidor LDS-ME principal e, caso seja desconectado de forma inesperada, o servidor de controle detecta a omissão de resposta da conexão e realiza a remoção do registro do componente.

Na avaliação, simulando as estações de trabalho em máquinas virtuais, o método resultou em um processo de descobrimento com tempo menor do que um segundo utilizando a biblioteca open62541, o quê inclui a estação de trabalho encontrar os servidores LDS-ME, registrar-se e obter informação dos outros servidores OPC UA. O resultado é relativamente rápido, em comparação com a implementação em Java, utilizando o Eclipse Milo, o qual obteve tempo de sete segundos.

2.2.2 Componentes I4.0

Para que dispositivos de campo ou módulos de processo possam ser automaticamente integrados no sistema de produção, devem dispor de modelos de descrição e capacidades que sigam um meta-modelo pré-definido que possa ser reconhecido pelos atores do sistema. Para isso, é especificado em Deutsches Institut für Normung (2016) o conceito de ativos de produção, *assets* na literatura em inglês, os quais são recursos que possuem algum valor para uma organização, representando "[. . .] um artefato que é especificamente destinado a executar um determinado papel em um determinado sistema" (p. 10).

O modelo de referência RAMI4.0 classifica os ativos organizacionais em três submundos: o mundo humano, o mundo das informações, composto por modelos, estados e arquivos, e o mundo físico. O papel principal da I4.0 é representar virtualmente um ativo técnico do mundo físico no mundo das informações, entretanto, um ativo pode, também, ser não-físico, como uma ideia, um programa de software, um arquivo ou um recurso, e, ainda ativos podem ser combinados para criar outros novos, com diferentes propriedades.

Os ativos são classificados em termos de sua capacidade de comunicação (nenhuma, passiva, ativa ou compatível com o sistema) e em termos da sua representação no sistema de informação (desconhecido, anônimo, conhecido ou entidade). Denomina-se *componentes I4.0* ativos que podem ser classificados com capacidade de comunicação compatível com o sistema e representados ou como individualmente conhecidos, ou como entidades, em que, nesse último caso, são

representados e administrados no mundo das informações (DEUTSCHES INSTITUT FÜR NORMUNG, 2016).

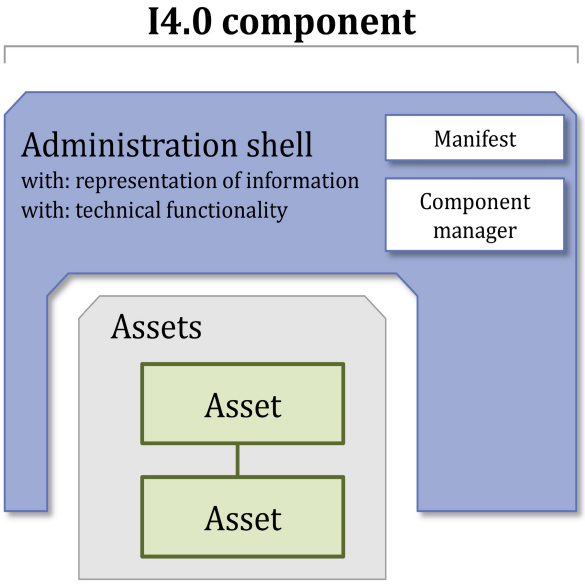
2.2.3 Asset Administration Shell

Dito que um componente I4.0 requer compatibilidade de comunicação, ou seja, integrabilidade com o sistema, o ativo deve ser encapsulado por uma representação virtual e ativa no sistema I4.0. Dessa forma, é recomendado pela referência RAMI4.0, em Deutsches Institut für Normung (2016), o uso de uma estrutura denominada Asset Administration Shell (AAS), isso é, uma *carcaça* de administração de ativos, cuja estrutura básica é caracterizada por um cabeçalho e um corpo de informação em referência ao domínio de fábrica digital, conforme especificado em International Electrotechnical Commission (2020).

O cabeçalho de informação encapsula a identificação do AAS em questão e dos ativos gerenciados por ele, informando como utilizá-lo, tal que deve, também, incluir as propriedades de classificação de nível requisitado de segurança para comunicação com o ativo. O corpo de informação possui as propriedades, vistas suportadas, serviços e referências, sendo o principal veículo de informação para com o ativo.

A Figura 13 mostra a representação da composição de dois ativos em um só, encapsulado por AAS, resultando em um componente I4.0.

Figura 13 – Representação de múltiplos ativos em AAS



Fonte: Deutsches Institut für Normung (2016, p. 39).

O campo *Manifest* possui as propriedades e funções do ativo resultante através de elementos de dados de condições gerais de negócios, construção, funcionalidade, localização e capacidade, ou performance.

O campo *Component manager* é uma unidade funcional que encapsula as

funções de administração do ativo, tal que seus serviços são providos pela API orientada a serviço do sistema I4.0.

Especifica-se em Alemanha (2020) os detalhes de modelagem de AAS, bem como a construção de meta-modelos, identificação de elementos, eventos e mapeamento para formato de dados. Quanto às fases do ciclo de vida de AAS, tem-se a definição de tipo e instância de elementos, em que o tipo engloba a fase de desenvolvimento e concepção do ativo, já a instância, caracteriza a fase de produção dos ativos, com base nas informações de tipo, e, também, seu uso e manutenção.

Especifica-se em OPC Foundation (2022a), também denominado norma IEC 62541-100, a extensão da modelagem de informações em OPC UA associada à dispositivos, abordando a sua modelagem base, modelos de comunicação e de integração com a máquina hospedeira. Na especificação, o tipo de dado DeviceType é definido para representar dispositivos de campo, incluindo as informações de vendedor, como código de produto, versão de software e número de série, informações de saúde do dispositivo e informações de protocolo de comunicação.

A Figura 14 mostra o mapeamento das informações de AAS para a especificação IEC 62541-100 em OPC UA, em que os elementos de dados são representados em OPC UA como variáveis e os serviços são representados como métodos. Nota-se que nem todos os elementos de AAS podem ser mapeados, por não haver correspondência na especificação IEC 6241-100.

Os padrões de informação em OPC UA têm como propósito prover acesso às informações dos dados de administração dos componentes e compartilhar os dados entre as operações de produção. Entretanto, a serialização das informações de AAS de dispositivos utiliza formatos de dados como XML e JavaScript Object Notation (JSON), conforme especificado em Alemanha (2020), definindo o formato de troca de dados entre as camadas de produção.

Figura 14 – Mapeamento de informações de AAS para OPC UA DeviceType

AAS		Device Type (from IEC 62541-100)
Header	↔	Object (ParameterSet)
Data Element (ManufacturerId)	↔	Variable(ManufacturerId)
Data Element (ModelId)	↔	Variable(ModelId)
Data Element (SerialNumber)	↔	Variable(SerialNumber)
Data Element (<ProfileId>)	↔	Variable(<ProfileId>)
Data Element (SecurityClass)	X	
Body	X	
Collection of Data Elements (Device Parameters)	↔	Object (ParameterSet)
Data Element (<ParameterIdentifier>)	↔	Variable (<ParameterIdentifier>)
Collection of Data Elements (Device Properties)	X	
Data Element (SerialNumber)	↔	Variable (SerialNumber)
Data Element (RevisionCounter)	↔	Variable (RevisionCounter)
Data Element (Manufacturer)	↔	Variable (Manufacturer)
Data Element (Model)	↔	Variable (Model)
Data Element (DeviceManual)	↔	Variable (DeviceManual)
Data Element (DeviceRevision)	↔	Variable (DeviceRevision)
Data Element (SoftwareRevision)	↔	Variable (SoftwareRevision)
Data Element (HardwareRevision)	↔	Variable (HardwareRevision)
Data Element (DeviceClass)	↔	Variable (DeviceClass)
Data Element (DeviceHealth)	↔	Variable (DeviceHealth)
Collection of Data Elements (Device Support Information)	X	
Data Element (DeviceTypeImage)	↔	Object (DeviceTypeImage)
Data Element (Documentation)	↔	Object (Documentation)
Data Element (ProtocolSupport)	↔	Object (ProtocolSupport)
Data Element (ImageSet)	↔	Object (ImageSet)
Collection of Services	↔	Method Set
Service (<MethodIdentifier>)	↔	Method (<MethodIdentifier>)
Service (Lock)	↔	Lock
View (<GroupIdentifier>)	↔	FunctionalGroupType (<GroupIdentifier>)
View (Identification)	↔	Identification
Reference	X	

Fonte: Alemanha (2017, p. 42).

2.2.4 Skills

A arquitetura de um sistema P&P exige que um gerenciador de dispositivos seja capaz de avaliar as capacidades dos componentes conectados ao sistema, como definido na quarta etapa do modelo de implementação proposto em Alemanha (2017). No contexto de componentes P&P, define-se *capacidade* como a habilidade de executar uma ação específica que cause um efeito específico, físico ou não, porém, não envolvendo os parâmetros e passos de execução para realizar a ação (PROFANTER, 2021).

Onori, Barata e Frei (2006) definem o conceito de sistemas de montagem evoluíveis, Evolvable Assembly Systems (EAS), como forma de preencher os requisitos de produção adaptativa, descrevendo os sistemas como simplificados e orientados aos processos. A evolução dos sistemas de montagem é baseada em elementos modulares, reconfiguráveis e de tarefas específicas, cujas capacidades são descritas por meio de *skills* de cada módulo, também abordado em Ferreira e Lohse (2012), tal que a interação dos subsistemas de produção resultem em funcionalidade complexas, geradas por meio da composição de skills.

Em 2006 é iniciado um projeto, financiado pela União Europeia, para facilitar e desenvolver as técnicas de reconfiguração dinâmica de processos complexos de produção com o uso de skills, denominado Skill-Based Inspection and Assembly for Reconfigurable Automation Systems (SIARAS). Malec et al. (2007) e Angelsmark et al. (2007) discutem o uso de skills como ontologias para reconfiguração de sistemas de produção, em que é proposto o uso de um modelo hierárquico de skills abstratas para implementar um servidor de integração de skills envolvidas no sistema. A classificação e mapeamento de skills a partir de programação orientada à tarefas é abordada por Backhaus, Ulrich e Reinhart (2014).

Björkelund et al. (2012) introduzem o uso de semânticas com skills para aumentar o nível de inteligência de roboôs de sistemas de automação, expandindo o modelo de produto-processo-recurso (PPR) (CUTTING-DECELLE et al., 2007), com o elemento *Skills*, que integra os outros três por meio da capacidade de integrar o conhecimento proporcionado pelos recursos para uso em diferentes processos e produtos. Nesse contexto, define-se outros três conceitos para sistematização de robôs: movimento, ação e tarefa. Uma skill é definida como uma MEF que executa cada um dos movimentos por meio de transições de estados, os quais são superconjuntos das tarefas envolvidas.

Ainda no contexto de PPR com skills, Pfrommer, Schleipen e Beyerer (2013) definem recursos como uma entidade de software ou hardware envolvida em um processo de execução, tal que processos são também denominados *skills abstratas*, independente de produtos e recursos, ou seja, são genéricos. As skills são então

definidas como a habilidade de um recurso para executar um processo, porém, diferente de processos, possuem mais informações. Os autores trazem, também ao modelo PPR, o conceito de tarefas, definidas pela aplicação de uma skill em um determinado tipo de produto com uma saída desejada, relacionando produto com skill.

Pfrommer et al. (2015) também definem skills como MEFs e propõem o uso do padrão OPC UA como forma de orquestrar sistemas de manufatura reconfiguráveis com uma arquitetura SOA, tal que os serviços são expostos como ações, que caracterizam a transformação de um produto por meio de uma skill. Os autores utilizam encapsulamento de componentes de produção para a execução das suas skills, em que os componentes são agregados a diferentes camadas de MES, que podem gerenciar os estados de execução das skills dos dispositivos, interconectados por um sistema superior de gerenciamento de ativos.

2.2.5 Implementação em OPC UA

O modelo de implementação de P&P em componentes industriais, representado no diagrama da Figura 10, pode ser descrito em termos do padrão OPC UA utilizando os conceitos abordados até aqui, seja o serviço de configuração de rede, responsável pelo descobrimento, como colocado na Seção 2.2.1. Seja, ainda, validação e autenticação de dispositivos, clientes e servidores, feitas de acordo com a especificação do modelo de segurança em OPC Foundation (2018b).

Descrever skills como MEFs possibilita implementá-las como Programas OPC UA, de acordo com a Seção 2.1.2.4, segundo proposto por Dorofeev e Zoitl (2018), que utilizam o modelo de programas como base para os serviços de controle de componentes, colocando skills e serviços como termos equivalentes, baseado em SOA. A flexibilidade dos programas em OPC UA permite executar skills atômicas e complexas através da extensão do espaço de estados da MEF, como uma skill de abertura e fechamento de válvula, que pode ser implementada pelos métodos `start()` e `stop()`. A composição de skills pode ser feita em OPC UA a partir das instâncias de dois ou mais clientes em um servidor compositor, que integra duas ou mais funcionalidades em uma skill complexa.

Profanter et al. (2019) apresentam uma modelagem de manipuladores robóticos e ferramentas por meio de skills, independente do hardware, provendo uma interface padronizada utilizando programas OPC UA, tal que as informações são descritas em um NodeSet, um arquivo que descreve todo o conjunto de nós envolvidos na aplicação. Os autores definem o tipo de objeto SkillType como um subtipo de OPC UA ProgramStateMachineType, estendendo-o com um conjunto de propriedades bases, como o parâmetro de nome para a skill. Define-se, também, uma interface ISkillControllerType, a qual agrupa todas as skills de um componente em um objeto Skills, permitindo o descobrimento genérico de skills em diferentes dispositivos.

A implementação em OPC UA é feita utilizando a biblioteca open62541 (Seção 2.1.3), porém com a linguagem C++, orientada a objetos, tal que a modelagem de informação é descrita por NodeSets em arquivos XML, como a definição de SkillType mostrada na Listagem 2.1. Os tipos de dados, como SkillType, são definidos no namespace `https://fortiss.org/UA/Device`, que estende o NodeSet OPC UA for Devices Integration (OPC UA DI). A composição de skills é demonstrada por meio da skill PickPlaceSkill, composta por MoveSkill e GripperSkill, que utilizam um manipulador robótico e um atuador do tipo garra.

Listagem 2.1 – Descrição em XML do tipo de objeto SkillType

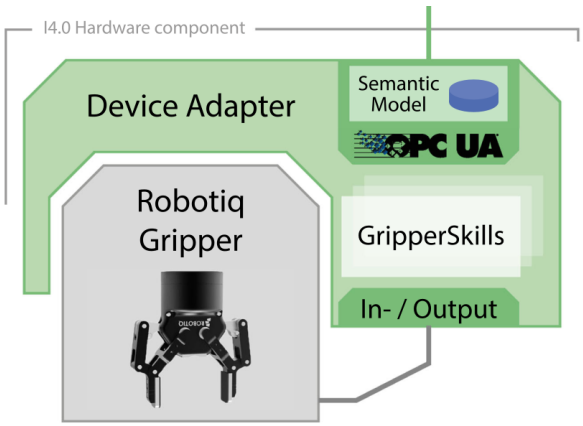
```
<ObjectType SymbolicName="FOR_DI:SkillType" BaseType="
    ↪ OpcUa:ProgramStateMachineType" IsAbstract="true">
    <Description>A skill type</Description>
    <Children>
        <Property SymbolicName="FOR_DI:Name" DataType="OpcUa:String
            ↪ " ValueRank="Scalar" ModellingRule="Mandatory">
            <Description>Name of the skill</Description>
        </Property>
    </Children>
</ObjectType>

<ObjectType SymbolicName="FOR_DI:GripperSkillType" BaseType="
    ↪ FOR_DI:SkillType" IsAbstract="true">
    <Description>A gripper skill type</Description>
</ObjectType>
```

Profanter (2021) utiliza os mesmos conceitos e abordagens para integrar componentes I4.0 em sistemas robóticos, trazendo, ainda, o uso de Device Adapters, Figura 15, uma implementação de AAS, que provê a funcionalidade de um dispositivo, adaptando sua interface proprietária, em um modelo de skills OPC UA.

Uma aplicação de um sistema robótico P&P genérico, composto de skills em OPC UA, é implementada por Profanter et al. (2021), em que os autores apresentam uma arquitetura para descobrimento de componentes I4.0, avaliação e composição de skills, representada na Figura 16. Os componentes, de hardware e de software, são conectados a mesma rede OPC UA, descobertos e orquestrados pelo componente MES, que realiza a detecção e possibilita a execução de skills.

Os autores utilizam um sistema robótico de produção composto por um braço robô e por diferentes ferramentas, tal que ao robô são atribuídas as skills de movimento, às ferramentas de garra as skills de *pegar* e *colocar* e, ainda, utiliza-se um componente de troca de ferramentas, possibilitando ao robô selecionar uma ferramenta, também por meio de skills. As skills de *mover*, *pegar* e *colocar* são compostas em uma única

Figura 15 – Device adapter



Fonte: Profanter (2021, p. 82).

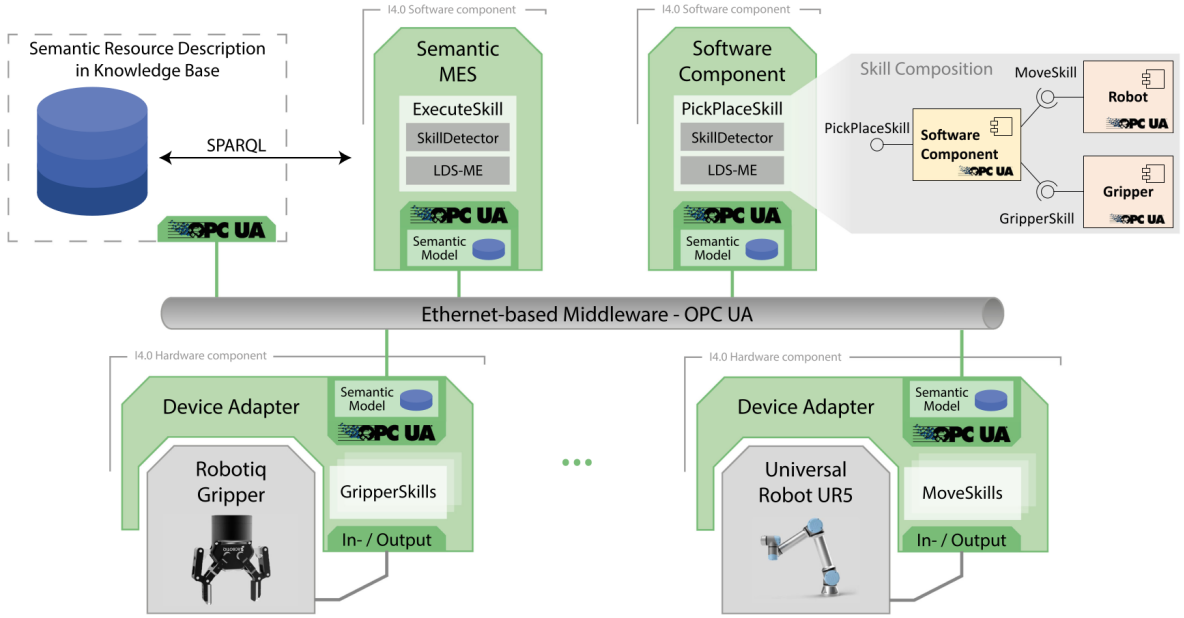
skill denominada PickAndPlace, conforme mostrado no diagrama de relações de skills da Figura 17.

Quanto aos adaptadores de interface de dispositivos, utiliza-se um microcontrolador embarcado, do tipo TinyPICO, baseado na plataforma ESP32, com o sistema operacional FreeRTOS, para integrar os dispositivos de ferramentas à rede OPC UA, o que é possibilitado pelo uso embarcado da biblioteca open62541[4]. O dispositivo apresenta um resultado de tempo de 9 segundos desde a inicialização do microcontrolador até a conclusão da detecção de skills.

Uma descrição completa de todo o trabalho é feita por Profanter (2021), em sua tese de doutorado, a qual servirá de base e literatura principal para a execução deste trabalho de conclusão de curso. Todo o código da aplicação está disponível em um repositório do GitHub[5], sob propriedade do instituto fortiss GmbH.

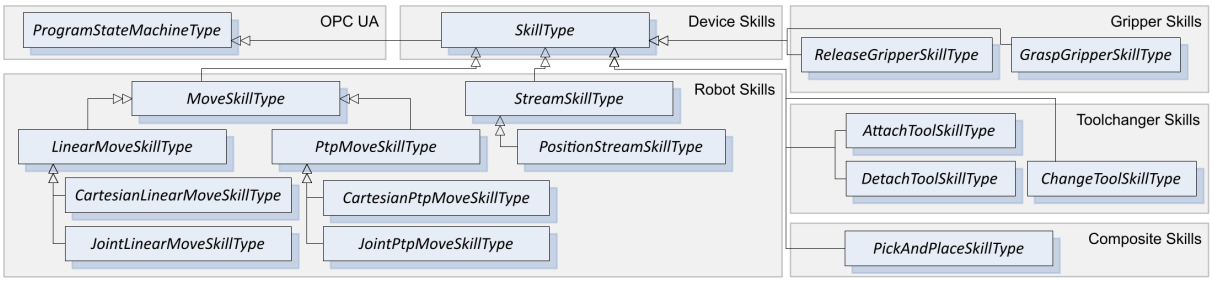
[4] https://github.com/Pro/open62541-esp32
[5] https://github.com/opcua-skills/plug-and-produce

Figura 16 – Exemplo de arquitetura de um sistema P&P com composição de skills em OPC UA



Fonte: Profanter et al. (2021, p. 130).

Figura 17 – Exemplo de modelagem de skills em OPC UA



Fonte: Profanter et al. (2021, p. 134).

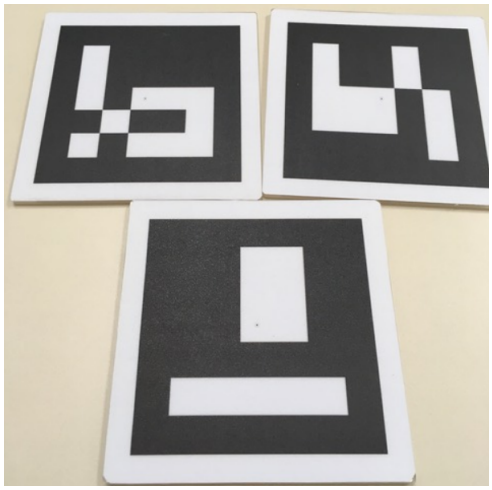
2.3 MARCADORES FIDUCIAIS

O uso de câmeras tem sido fortalecido por pesquisas acerca da visão computacional, que faz o uso de processamento de imagens para identificar objetos específicos. Avaliações preliminares indicam que algumas dessas técnicas envolvendo triangulação ou multilateração de sinais de rádio-frequência ou ultrassom, sensores inerciais e processamento de imagens são capazes de fornecer dados dinâmicos de posição com erros da ordem de poucos milímetros (AMSTERS et al., 2019).

Originalmente desenvolvidos para aplicações de realidade aumentada, sistemas ópticos com marcadores fiduciais têm se tornado cada vez mais populares em tarefas de robótica móvel, caracterizados por marcas codificadas planas e quadradas, que são posicionadas no ambiente ou no objeto a ser rastreado (CELOZZI et al., 2010; WAGNER; SCHMALSTIEG, 2007).

Garrido-Jurado et al. (2015) apresentam um dicionário de marcadores fiduciais denominado ArUco, caracterizados por marcadores quadrados com uma ampla borda em preto e uma matriz binária central que determina o seu identificador numérico, conforme mostrado na Figura 18, um conjunto de marcadores ArUco de tamanho 5x5, ou seja, a matriz é composta por 25 bits, que determinam o ID de cada marcador.

Figura 18 – Marcadores fiduciais ArUco



Fonte: autor (2022).

Romero-Ramirez, Muñoz-Salinas e Medina-Carnicer (2018) desenvolveram uma biblioteca[6], em C++, para a detecção e cálculo de pose de marcadores fiduciais em uma imagem, com tempo de processamento de até 2,75 ms para imagens com resolução de 2160p.

Entretanto, a foto resultante de uma câmera possui distorções resultantes da curvatura e abertura da lente, logo, a fim de determinar um ponto geométrico na imagem, é necessário utilizar os parâmetros de calibração da câmera, respectivamente,

[6] https://www.uco.es/investiga/grupos/ava/node/26

os coeficientes de distorção, que lineariza a grade da imagem, e a matriz de câmera, que mapeia os pontos da imagem para medidas espaciais, em mm (LUHMANN et al., 2019).

$$\text{Coeficientes de Distorcao} = (k_1 \quad k_2 \quad p_1 \quad p_2 \quad k_3) \tag{1a}$$

$$\text{Matriz de Camera} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \tag{1b}$$

As os coeficientes das Equações 1a e 1b podem ser determinados através de um método proposto em OpenCV (2022). Esses coeficientes são necessários para o uso da biblioteca ArUco, desenvolvida por Romero-Ramirez, Muñoz-Salinas e Medina-Carnicer (2018), a fim de mapear em coordenadas espaciais a pose de marcadores fiduciais em uma imagem.

3 DESCRIÇÃO E REQUISITOS DO SISTEMA

Neste Capítulo é proposto um estudo de caso de um sistema de produção que pode ser automatizado por meio da introdução de componentes com funcionalidade P&P. Entretanto, a abordagem que será utilizada na metodologia de projeto de implementação pode ser estendida para diferentes cenários de produção que envolvem fábricas adaptativas. Este trabalho apresenta um demonstrador para a solução com semânticas de *skills*, ou, em outros termos, uma prova de conceito.

3.1 ESTUDO DE CASO

As instalações de sistemas de produção são compostas por diferentes elementos físicos como máquinas, produtos e sensores. Como forma de identificar ou rastrear determinados objetos, ou, até mesmo, orientar sistemas robóticos, pode-se utilizar identificadores anexados a cada um desses objetos.

Considera-se, então, um cenário com a presença de elementos que devem ser periodicamente identificados e localizados. Uma solução é atribuir a cada um desses elementos um identificador unívoco, ou seja, um ID, porém, que deve ser visualmente identificável e localizável, como no caso, através de um marcador fiducial, descrito na Seção 2.3.

Propõe-se, então, distribuir câmeras pelo ambiente que, periodicamente, são processadas, retornando quais marcadores foram encontrados e a sua posição relativa, o que é possibilitado pela implementação da biblioteca ArUco (GARRIDO-JURADO et al., 2015; ROMERO-RAMIREZ; MUÑOZ-SALINAS; MEDINA-CARNICER, 2018). Entretanto, nesse modelo é necessário o pré-conhecimento da pose das câmeras em relação a um sistema coordenado fixo M-xyz qualquer, denominado *map*.

Utilizando a biblioteca ArUco, atribuí-se a cada câmera C_i a posição $\mathbf{f_j^{C_i}}$ de cada marcador fiducial F_j em relação ao sistema coordenado da câmera em questão. Se $\mathbf{o_{C_i}^{M}}$ descreve a origem da câmera C_i no sistema coordenado M, *map*, e $\mathbf{R_{C_i}^{M}}$ a matriz de rotação da câmera no sistema coordenado M, define-se a matriz de transformação homogênea de C_i para M.

$$\mathbf{A_{C_i}^{M}} = \begin{bmatrix} \mathbf{R_{C_i}^{M}} & \mathbf{o_{C_i}^{M}} \\ \mathbf{0^T} & 1 \end{bmatrix} \tag{2}$$

Logo, a posição de um marcador fiducial F_j em relação ao sistema coordenado M, visto pela câmera C_i, ou seja, $\mathbf{f_j^{M_{C_i}}} = [x_j^{M_{C_i}} \quad y_j^{M_{C_i}} \quad z_j^{M_{C_i}}]^T$, é calculada através de:

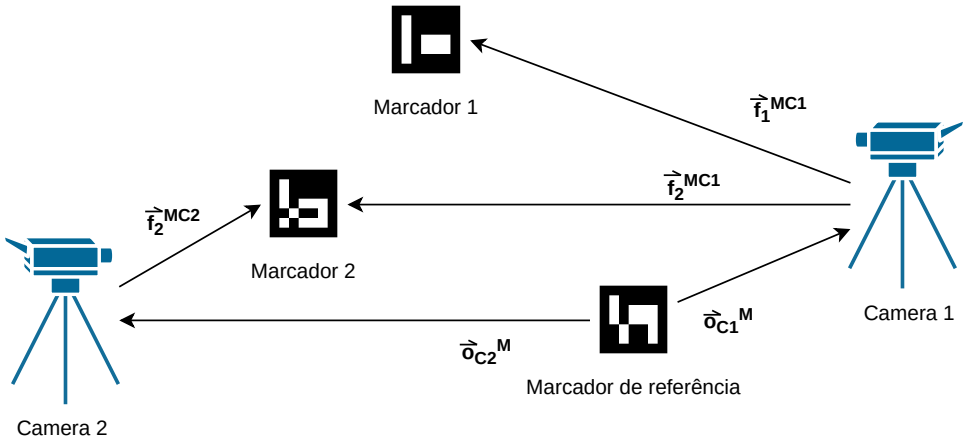
$$\tilde{\mathbf{f}}_\mathbf{j}^{\mathbf{M_{C_i}}} = \mathbf{A_{C_i}^{M}} \cdot \tilde{\mathbf{f}}_\mathbf{j}^{\mathbf{C_i}} \tag{3}$$

em que $\tilde{\mathbf{f}}_{\mathbf{j}}^{\mathbf{M_{C_i}}} = [x_j^{M_{C_i}} \quad y_j^{M_{C_i}} \quad z_j^{M_{C_i}} \quad 1]^T$ é um vetor auxiliar que determina as coordenadas xyz do marcador F_j. Se um marcador fiducial é visto por n_C câmeras, a sua posição unívoca é determinada pela média das suas posições em relação a M.

$$\mathbf{f}_{\mathbf{j}}^{\mathbf{M}} = \frac{\sum_{i=1}^{n_C} \mathbf{f}_{\mathbf{j}}^{\mathbf{M_{C_i}}}}{n_C}, \qquad n_C \geq 1 \tag{4}$$

A Figura 19 mostra um cenário de identificação de marcadores fiduciais, tal que utiliza-se um marcador como referência, ou seja, representando o sistema coordenado M. O Marcador 1 é identificado apenas pela Câmera 1, logo, sua posição é $\mathbf{f}_{\mathbf{1}}^{\mathbf{M}} = \tilde{\mathbf{f}}_{\mathbf{1}}^{\mathbf{M_{C_1}}}$. Já o Marcador 2 é identificado pelas Câmeras 1 e 2, portanto, sua posição é dada pela média das posições fornecidas por cada uma das câmeras, segundo a Equação 4.

Figura 19 – Exemplo de identificação de marcadores



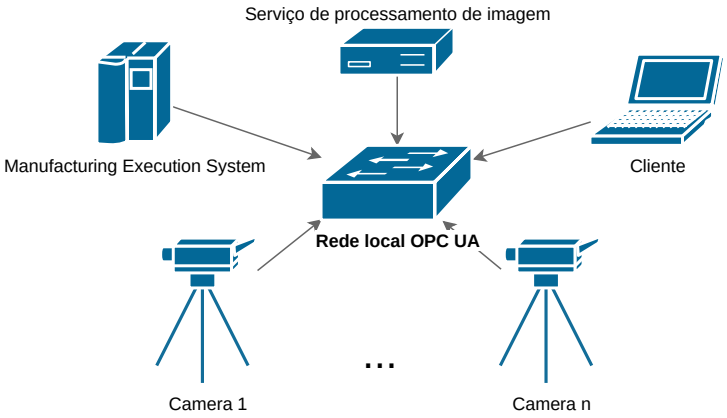
Fonte: autor (2022).

Portanto, os dados de interesse, fornecidos pelo sistema, são caracterizados por quais marcadores estão, em um determinado momento, dentro, e visíveis, do ambiente e qual a sua posição em relação a um sistema coordenado pré-definido. Dessa forma, o sistema pode operar periodicamente realizando verificações em todas as câmeras presentes no sistema, que serão detectadas por meio das técnicas de P&P apresentadas na Seção 2.2.

Quanto aos componentes do sistema, propõe-se uma infraestrutura de rede local, composta por um servidor principal, que atua como um sistema MES, através do qual, é possível interagir, realizando leituras e requisições, conforme representado na Figura 20. A identificação de marcadores é divida em dois grupos de componentes, sendo um conjunto de n_C câmeras, com a funcionalidade de retornar imagens, e um componente de software para o processamento das imagens, que realiza a identificação de marcadores fiduciais em imagens.

Considerando a abstração de um sistema P&P, os complentes I4.0 devem ser descritos de forma genérica, ou seja, um dispositivo de câmera pode corresponder a tanto a webcam de um laptop, quanto uma câmera profissional, com interface de

Figura 20 – Componentes do sistema



Fonte: autor (2022).

conexão à Internet, encapsulada por AAS (ALEMANHA, 2017; ALEMANHA, 2020). Um terceiro componente de software deve, então, realizar a composição das habilidades providas pelas câmeras e pelo processador de imagens, resultando na funcionalidade de procurar marcadores fiduciais.

3.2 CASO DE USO

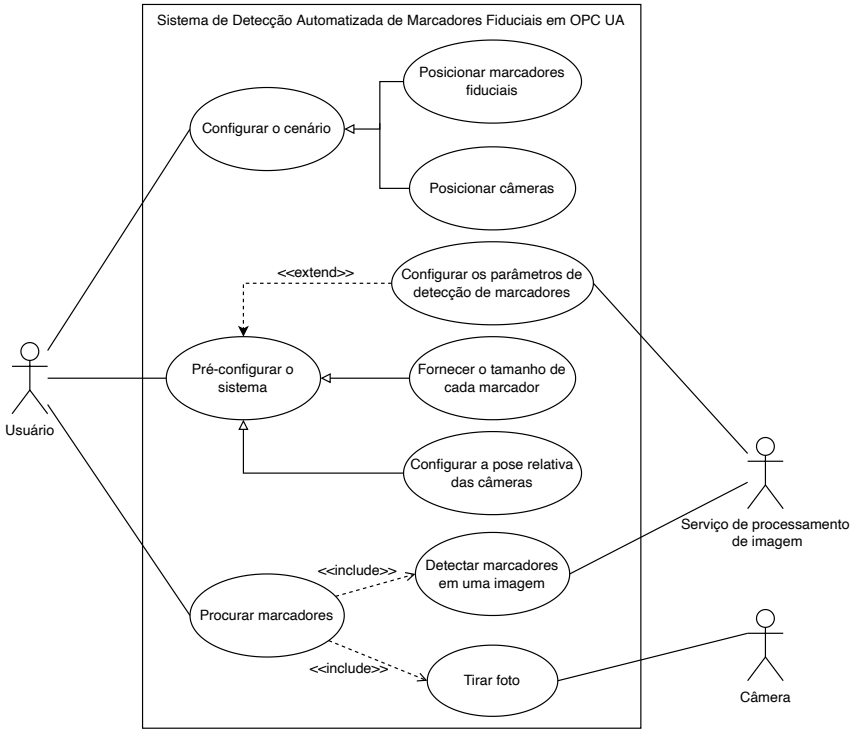
Um usuário do sistema é colocado como aquele que, possuindo um conjunto de marcadores fiduciais, de diferentes tamanhos, os espalham sistematicamente por um ambiente. Ainda, o usuário é responsável por instalar as câmeras no local, tendo conhecimento de sua pose relativa, ou utilizando algum serviço para determinar esses parâmetros e configurar os dispositivos. A Figura 21 mostra o diagrama de caso de uso para o sistema.

É função do usuário pré-configurar o cenário, devendo fornecer ao sistema o conjunto de marcadores com seus respectivos tamanhos, bem como garantir a configuração das câmeras. O usuário pode, ainda, alterar os valores padrões de detecção de marcadores no serviço de processamento de imagem. Através do software, o usuário solicita o processo de procura, esperando como resultado a posição relativa de cada um dos marcadores encontrados pelas câmeras.

3.3 LEVANTAMENTO DE REQUISITOS

Fundamental para projetos de software, requisitos são descritos pela especificação de uma característica ou propriedade que um sistema ou componente deve assumir para atingir o objetivo do projeto, além de determinar suas restrições de operação. A etapa de levantamento de requisitos lista um conjunto de metas que o projeto final deve cumprir, essencial para a fase de avaliação de resultados,

Figura 21 – Diagrama de caso de uso da detecção automatizada de marcadores fiduciais



Fonte: autor (2022).

embora este trabalho se trata de uma prototipagem, para estudar as alternativas de interface, problemas de comunicação e avaliação de desempenho da implementação de componentes P&P utilizando o padrão OPC UA (PRESSMAN, 2011).

3.3.1 Requisitos funcionais

O intuito dos requisitos funcionais é apontar como o sistema deve se comportar em situações específicas, ou ainda, o que o software deve ter em termos de tarefas ou serviços de usuário (VAZQUES; SIMÕES, 2016).

RF1 O descobrimento é gerenciado pelo OPC UA e acontece logo após a conexão de novos dispositivos à rede.

RF2 Os componentes podem ser configurados para serem avisados quando outros dispositivos são conectados.

RF3 Os dispositivos ficam disponíveis imediatamente após a conexão.

RF4 A desconexão de um dispositivo não trava a execução do sistema.

RF5 As exceções disparadas ao longo da execução são tratadas pelo sistema.

RF6 Todo dado é válido e a falta de pré-configuração do sistema não implica que o modelo é não-operável, mas se comportará dentro de suas capacidades. Por exemplo, uma câmera não calibrada pode, mesmo assim, identificar marcadores fiduciais, mas não determinar a sua posição.

RF7 O sistema pode ser executado de forma distribuída.

3.3.2 Requisitos não funcionais

As restrições gerais do sistema são descritas pelos requisitos não funcionais, que descrevem limitações de ordem geral aos requisitos funcionais, estabelecendo níveis de serviços esperados, também atrelado à qualidade do sistema (VAZQUES; SIMÕES, 2016). No escopo deste trabalho, os requisitos não funcionais estão diretamente relacionados com os requisitos de P&P descritos em Alemanha (2017) e com os conceitos sobre I4.0 abordados em (ALEMANHA, 2019), baseados em autonomia, interoperabilidade e sustentabilidade.

RNF1 A autonomia é garantida pela acessibilidade à infraestrutura digital.
RNF2 A interoperabilidade é garantida pelos padrões de semânticas.
RNF3 O sistema é descentralizado e os recursos são utilizáveis mesmo quando separados.
RNF4 O sistema pode ser composto por diferentes modelos de câmeras.
RNF5 O sistema é de plataforma independente.

3.3.3 Regras de negócio

As políticas de funcionamento do sistema é gerenciada pelas regras de negócio, que complementam os processos ligados ao domínio do software.

RN1 Uma variável numérica não configurada recebe o valor `NAN`, ou seja, *not a number*.
RN2 As coordenadas de um marcador com posição inválida recebem o valor `NAN`. Porém, se tiver a posição reconhecida por uma outra câmera, o marcador herda o valor da posição válida.

4 PROJETO E IMPLEMENTAÇÃO

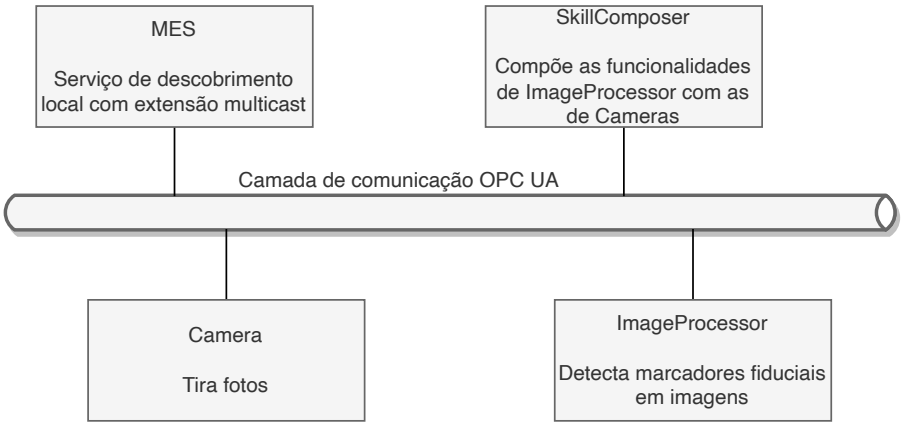
Primeiramente, o sistema proposto requer uma definição clara de seus atores e suas funcionalidades. Os quatro principais componentes para o problema descrito no capítulo anterior são genericamente definidos a seguir e resumidos graficamente na Figura 22.

1. Manufacturing Execution System (MES): provê o serviço de descobrimento para os demais servidores da rede OPC UA.
2. ImageProcessor: componente de software de processamento de imagens, que possui a skill de detectar a posição de marcadores fiduciais em uma imagem de entrada com base nos parâmetros de calibração da câmera.
3. Camera: dispositivos que possuem a skill de tirar fotos e guardam a imagem resultante em uma variável. As câmeras são configuradas com os parâmetros de calibração e pose em relação a um sistema coordenado de referência.
4. SkillComposer: componente de software que une a funcionalidade das câmeras com a funcionalidade do processador de imagens e resulta na skill de procurar marcadores.

Os componentes são descritos com base nos seus recursos e nas suas funcionalidades, seguindo o modelo de SOA, no processo denominado modelagem de informação, em OPC UA. Apesar da autonomia, os componentes dependem de alguns padrões de modelagens que regem a identificação das informações do dispositivo servidor, como no caso do uso de AAS, conforme descrito na Seção 2.2.3, que mapeia a camada de abstração para o tipo de dado DeviceType, em OPC UA.

Profanter et al. (2021) desenvolveram um conjunto de modelos e recursos orientados ao caso de uso de P&P e AF, compartilhados por todos os componentes que

Figura 22 – Definição e conexão dos atores do sistema



Fonte: autor (2022).

utilizam a funcionalidade, reutilizados e adaptados para este projeto. Nesse contexto, as próximas seções abordam a modelagem e implementação dos recursos compartilhados e específicos de cada componente.

4.1 MODELAGEM

A modelagem de informação é uma etapa comum a qualquer projeto de componentes OPC UA, independente da plataforma de desenvolvimento utilizada ao longo da implementação do projeto. Esse processo requer um entendimento do espaço de endereçamento do padrão OPC UA, resumido na Seção 2.1.2, bem como dos padrões, documentados em OPC Foundation (2022c).

As informações são representadas em OPC UA por NodeSets, isso é, um conjunto de nós que compõem o espaço de endereçamento do servidor, envolvem além de variáveis e métodos, as declarações de cada tipo de dado utilizado. Separa-se, aqui, as informações comuns aos componentes do sistema de suas informações específicas, agrupando-as conforme os seus casos de uso.

Conforme o processo de modelagem descrito por Profanter (2020), representado na Figura 9, cada NodeSet é primeiramente descrito por um arquivo XML, posteriormente interpretado pela ferramenta UA-ModelCompiler, resultando em um conjunto final de arquivos que especificam pontualmente o espaço de endereçamento do servidor através de NodeIds.

4.1.1 NodeSets de base

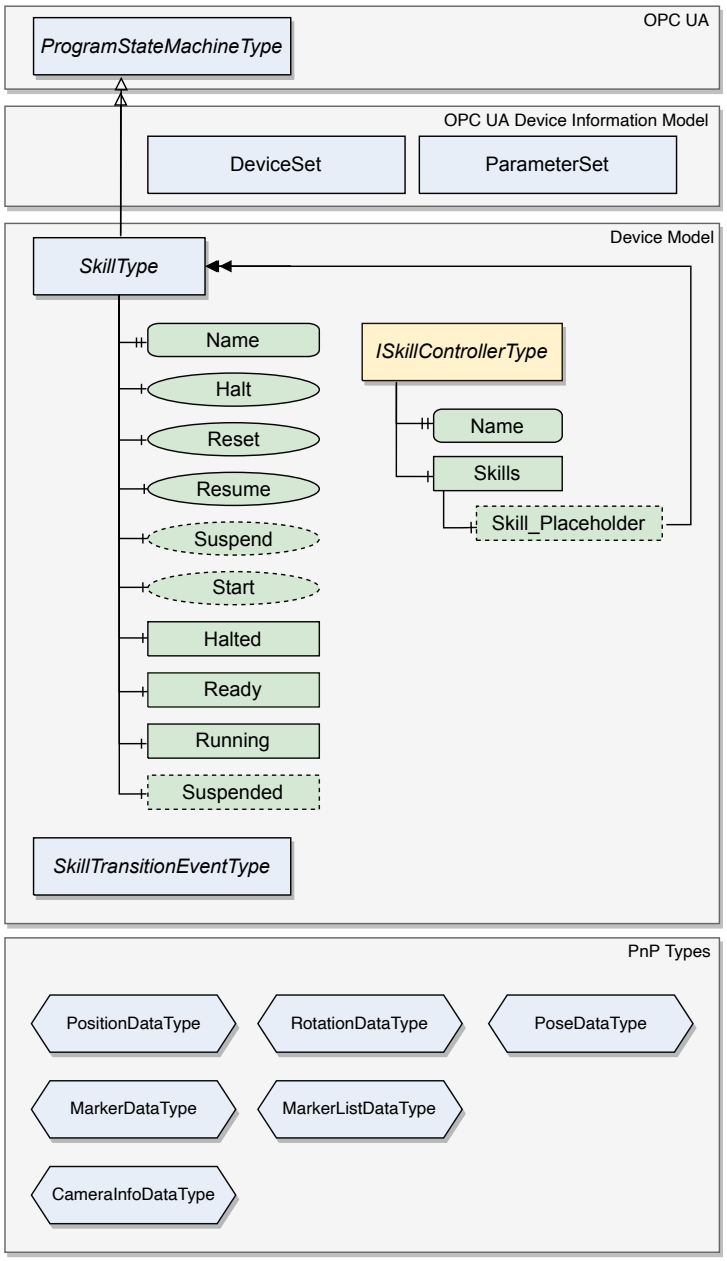
Para todo servidor OPC UA, a base da modelagem de informação é composta pelo NodeSet de índice zero, especificado na Parte 5 do padrão, que agrega os tipos de informação básicos para a construção de modelos mais complexos (OPC FOUNDATION, 2022c). A Parte 100 do padrão apresenta um conjunto de informações para modelagem de dispositivos, denominado Device Information Model, o qual é aplicado na construção de AAS em OPC UA (OPC FOUNDATION, 2022a; ALEMANHA, 2020).

Profanter et al. (2021) definem um NodeSet para modelagem de sistemas com semânticas baseadas em skills, denominado Device Model, conforme apresentado nas Seções 2.2.4 e 2.2.5, declarando o tipo de objeto abstrato SkillType, base para outra e qualquer skill, que herda do objeto de tipo ProgramSkillType, bem como o tipo de objeto SkillTransitionEventType para definir os eventos de transição entre os estados de uma skill. O NodeSet também dispõe de uma interface para dispositivos controladores de skill, como forma de auxiliar na busca de nós no servidor, identificando quais controladores esse determinado servidor possui.

As informações como marcadores fiduciais, pose, calibração de câmera, entre outras, levantam a necessidade de declarar um conjunto de tipos comuns de dados transferidos entre os componentes, definidos no NodeSet PnP Types. A Figura 23 resume os NodeSets de base e os principais tipos que os compõem.

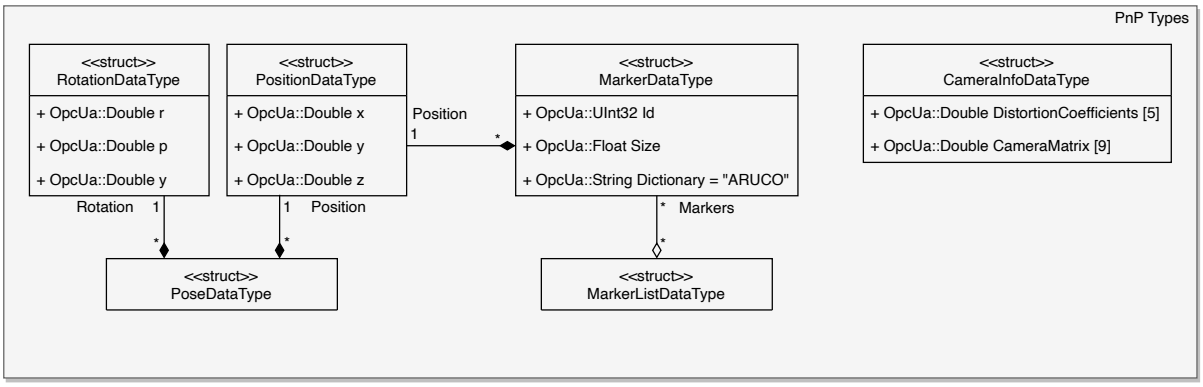
No NodeSet OPC UA Device Information Model, o objeto DeviceSet é definido como um ponto de início para localizar dispositivos em um servidor. ParameterSet é um objeto definido como um contêiner padrão para sistemas que possuem uma lista de parâmetros, como no caso de skills, que necessitam de parâmetros de entrada e saída de dados, denominados SkillParameters.

Figura 23 – NodeSets de base



Fonte: autor (2022).

Figura 24 – Diagrama de classes dos tipos de dados do NoseSet PnP Types



Fonte: autor (2022).

O NodeSet PnP Types declara apenas os tipos de dados utilizados no sistema, necessário para os diferentes servidores consigam interpretar as informações trocadas entre eles. O padrão OPC UA não deixa claro uma forma gráfica de representar estrutura de dados, logo, mostra-se na Figura 24 os diagramas de classe para os tipos do NodeSet.

A cada NodeSet é atribuído um Namespace, que caracteriza um identificador único para aquele NodeSet, representado por um endereço web, como exemplo, o Namespace do NodeSet base do padrão OPC UA é definido pelo endereço http://opcfoundation.org/UA/. Esse identificador reside nos servidores OPC UA e mapeia o índice do NodeSet para acesso às informações, de forma que um nó seja definido por esse índice e por mais um identificador, que pode ser numérico, binário ou textual.

Considera-se a Listagem 4.1, que exemplifica a definição de um tipo de dado que representa a informação de pose, composta pelos demais tipos de posição e rotação (Figura 24). O identificador do tipo é escrito seguido do identificador do NodeSet que o contém, no caso, PNPTYPES, ou, OpcUa, que se refere ao NodeSet base do padrão OPC UA.

Listagem 4.1 – Definição do tipo de dado de representação de pose em XML

```
1 <DataType SymbolicName="PNPTYPES:PositionDataType" BaseType="
    ↪ OpcUa:Structure" IsAbstract="false">
2     <Description>A representation of position in cartesian space</
        ↪ Description>
3
4     <Fields>
5         <Field Name="x" DataType="OpcUa:Double"/>
6         <Field Name="y" DataType="OpcUa:Double"/>
7         <Field Name="z" DataType="OpcUa:Double"/>
8     </Fields>
```

```
</DataType>

<DataType SymbolicName="PNPTYPES:RotationDataType" BaseType="
    ↪ OpcUa:Structure" IsAbstract="false">
    <Description>A representation of rotation in cartesian space</
       ↪ Description>

    <Fields>
        <Field Name="r" DataType="OpcUa:Double"/>
        <Field Name="p" DataType="OpcUa:Double"/>
        <Field Name="y" DataType="OpcUa:Double"/>
    </Fields>
</DataType>

<DataType SymbolicName="PNPTYPES:PoseDataType" BaseType="
    ↪ OpcUa:Structure" IsAbstract="false">
    <Description>A representation of pose in cartesian space,
       ↪ composed by position and rotation</Description>

    <Fields>
        <Field Name="Position" DataType="PNPTYPES:PositionDataType"
           ↪ />
        <Field Name="Rotation" DataType="PNPTYPES:RotationDataType"
           ↪ />
    </Fields>
</DataType>
```

A declaração dos tipos de dados é, em seguida, processada pela ferramenta UA-ModelCompiler, que tem como saída um conjunto de arquivos descritivos que determinam pontualmente o espaço de endereçamento do servidor. Dando continuidade ao exemplo da Listagem 4.1, a ferramenta gera o arquivo PnPTypes.NodeSet2.xml, que contém a definição dos nós, mostrado na Listagem 4.2.

Listagem 4.2 – Definição dos nós de tipos de pose no espaço de endereçamento

```
<UADataType NodeId="ns=1;i=15073" BrowseName="1:PositionDataType">
  <DisplayName>PositionDataType</DisplayName>
  <Description>A representation of position in cartesian space</
     ↪ Description>
  <References>
    <Reference ReferenceType="HasSubtype" IsForward="false">i=22</
       ↪ Reference>
  </References>
```

```
  <Definition Name="1:PositionDataType">
    <Field Name="x" DataType="i=11" />
    <Field Name="y" DataType="i=11" />
    <Field Name="z" DataType="i=11" />
  </Definition>
</UADataType>
<UADataType NodeId="ns=1;i=15004" BrowseName="1:RotationDataType">
  <DisplayName>RotationDataType</DisplayName>
  <Description>A representation of rotation in cartesian space</
    ↪ Description>
  <References>
    <Reference ReferenceType="HasSubtype" IsForward="false">i=22</
      ↪ Reference>
  </References>
  <Definition Name="1:RotationDataType">
    <Field Name="r" DataType="i=11" />
    <Field Name="p" DataType="i=11" />
    <Field Name="y" DataType="i=11" />
  </Definition>
</UADataType>
<UADataType NodeId="ns=1;i=15005" BrowseName="1:PoseDataType">
  <DisplayName>PoseDataType</DisplayName>
  <Description>A representation of pose in cartesian space,
    ↪ composed by position and rotation</Description>
  <References>
    <Reference ReferenceType="HasSubtype" IsForward="false">i=22</
      ↪ Reference>
  </References>
  <Definition Name="1:PoseDataType">
    <Field Name="Position" DataType="ns=1;i=15073" />
    <Field Name="Rotation" DataType="ns=1;i=15004" />
  </Definition>
</UADataType>
```

Observa-se que o valor de `ns`, igual a 1, é comum para os três tipos, pois se refere ao Namespace do NodeSet PNPTYPES.

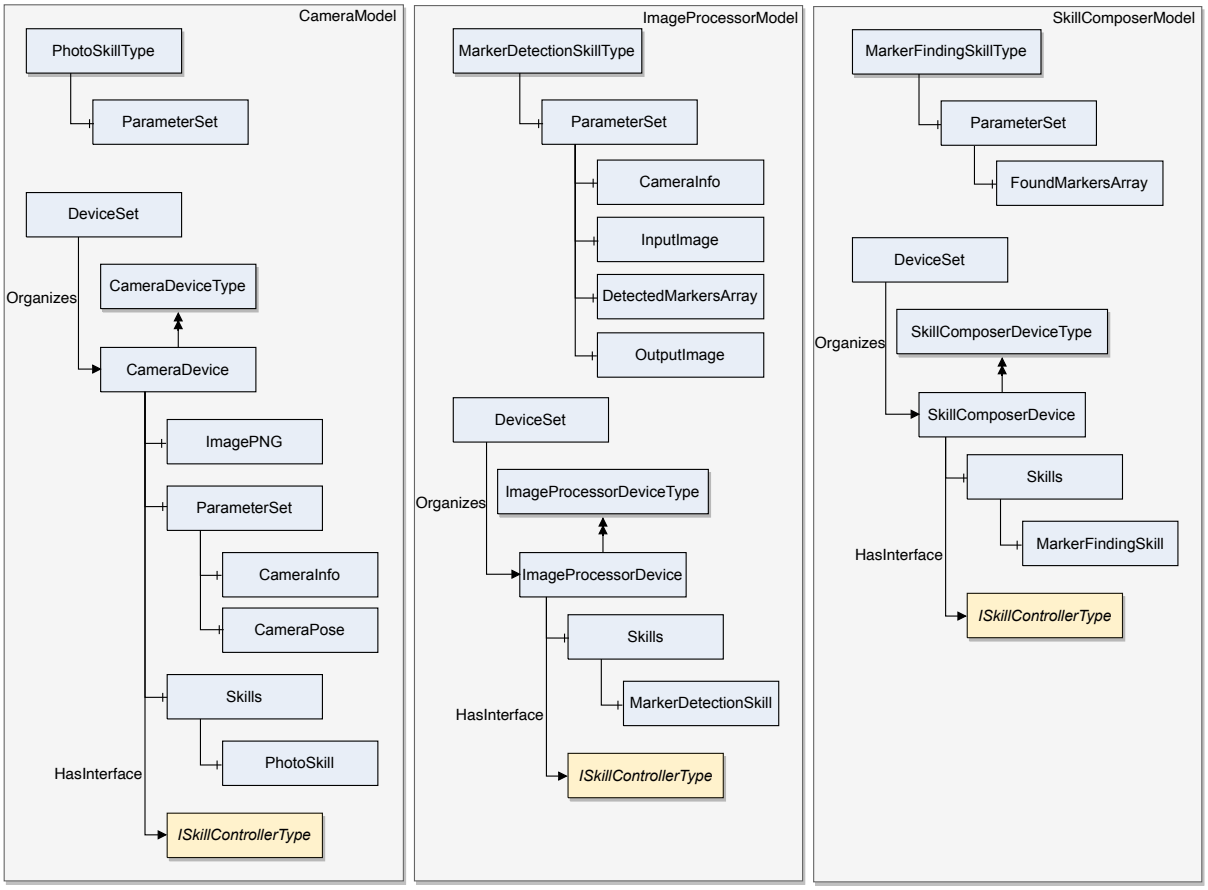
4.1.2 NodeSets específicos

A partir dos NodeSets de base, modelam-se as informações específicas para cada componente, ainda que alguns padrões permaneçam, como o objeto de dispositivo que, conforme o modelo de AAS, é organizado pelo objeto DeviceSet, que corresponde

ao conjunto de dispositivos existentes naquele servidor. Outra característica comum é a relação de interface de cada componente com ISkillControllerType, que carateriza o dispositivo como um controlador de skills e, logo, espera-se encontrar suas skills organizadas pelo objeto Skills.

A Figura 25 mostra os NodeSets para cada uma das entidades de componentes propostas, isso é, para a câmera, para o processador de imagens e para o compositor de skills. Apesar de não explícitas, existem relações entre os objetos mostrados e aqueles definidos nos NodeSets de base, como é o caso dos tipos de skills, que herdam de SKillType, do NodeSet DeviceModel.

Figura 25 – NodeSets específicos de componentes



Fonte: autor (2022).

O dispositivo de câmera possui um objeto do tipo padrão OPC UA ImagePNG, que corresponde à última foto processada pela câmera, ou seja, é alterado pela execução de PhotoSkill. A câmera possui também um objeto do tipo ParameterSet, definido no NodeSet OPC UA Device Information Model, que organiza os parâmetros de configuração de uma câmera, respectivamente, CameraInfo que é uma estrutura que corresponde aos valores de coeficientes de distorção e da matriz de câmera, e CameraPose, que determina a pose da câmera. Apesar de vazio, PhotoSkillType também possui um conjunto de parâmetros, podendo ser estendido.

No modelo do processador de imagens, ImageProcessorModel, os parâmetros

de MarkerDetectionSkillType são compostos por quatro variáveis, sendo duas de entrada e duas de saída. CameraInfo e InputImage são valores repassados da câmera para a skill de detecção de marcadores, que retorna um vetor de marcadores fiduciais detectados, DetectedMarkersArray, e uma imagem com a representação gráfica da detecção, OutputImage, do tipo ImagePNG.

Por ser um componente automatizado, o compositor de skills não requer nenhuma pré-configuração ou parâmetros de entrada, possuindo, apenas, a variável de saída FoundMarkersArray em seu modelo, SkillComposerModel, do mesmo tipo de DetectedMarkersArray, porém, composto e processado pelos dados dos marcadores encontrados por todas as câmeras presentes no sistema.

A ligação entre NodeSets é feita declarando os Namespaces dependentes no arquivo de modelagem XML, como no caso do modelo da câmera, em que a Listagem 4.3 mostra a declaração do Namespace PnPTypes como `PNPTYPES`, atribuído ao identificador `xmlns`, ou seja, XML Namespace, refente ao identificador único de Namespace https://pnp.org/UA/PnPTypes/. O mesmo processo é feito para os demais Namespaces dependentes, no caso, DeviceInformation, DeviceModel e, o Namespace próprio, Camera.

Listagem 4.3 – Conexão de Namespaces no arquivo XML de molagem

```
1  <ModelDesign
2    xmlns:DI="http://opcfoundation.org/UA/DI/"
3    xmlns:DEVICE="https://fortiss.org/UA/Device/"
4    xmlns:PNPTYPES="https://pnp.org/UA/PnPTypes/"
5    xmlns:CAMERA="https://pnp.org/UA/Camera/"
6
7    <Namespaces>
8      <Namespace Name="Camera" Prefix="Camera" XmlNamespace="https://
         ↪ pnp.org/UA/Camera/Types.xsd" XmlPrefix="Camera">https://
         ↪ pnp.org/UA/Camera/</Namespace>
9      <Namespace Name="PnPTypes" Prefix="PnPTypes" XmlNamespace="
         ↪ https://pnp.org/UA/PnPTypes/Types.xsd" XmlPrefix="
         ↪ PnPTypes" FilePath="../../../nodeset/PnPTypes/PnPTypes">
         ↪ https://pnp.org/UA/PnPTypes/</Namespace>
10     ...
11   </Namespaces>
12
13   ...
14 />
```

Além da declaração de cada Namespace, define-se cada um deles a partir da linha 7, no campo `Namespaces`, em que cada definição é referenciada com o valor do

identificador único de Namespace. Para os Namespaces dependentes, especifica-se o caminho relativo para o arquivo de saída de estensão `.xsd`, gerado pela ferramenta UA-ModelCompiler.

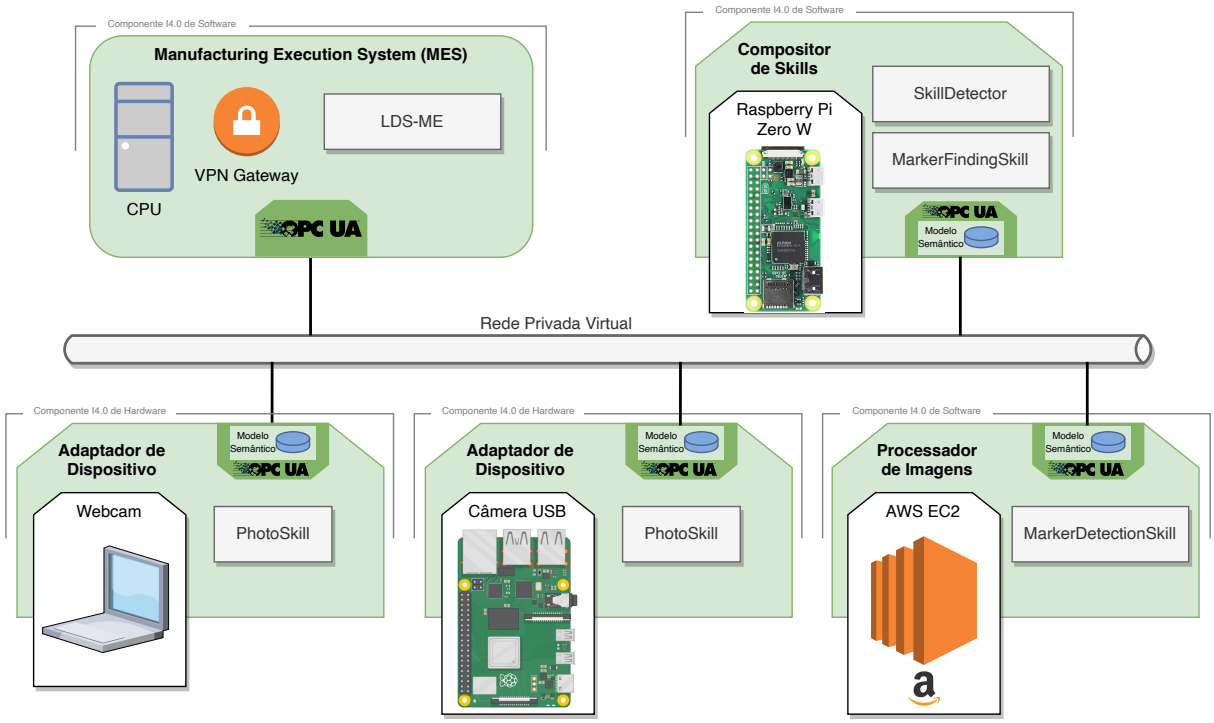
4.1.3 Arquitetura

OPC UA é um padrão cujo um dos requisitos é ser utilizado em sistemas distribuídos, de fato, serviços que exigem mais processamento podem ser alocados em servidores que não necessariamente se encontram na linha de produção. Uma das formas de garantir a distribuição eficiente do sistema é utilizar serviços em nuvem, como ofertado pela empresa Amazon Web Services (AWS), que oferece locação de infraestrutura computacional.

Este projeto propõe o uso de uma arquitetura distribuída em computadores pessoais, sistemas embarcados e servidores em nuvem, conectados a uma rede Virtual Private Network (VPN) utilizando a ferramenta WireGuard, integrado ao Linux desde a versão 5.6 do kernel, de março de 2020. A arquitetura do sistema, com todos os elementos participantes, é representada na Figura 26, onde o MES, executado em uma CPU, implementa, também, o serviço de conexão para a rede VPN — não implementado no servidor OPC UA.

Utiliza-se duas câmeras conectadas a um adaptador de dispositivo, uma webcam integrada em um computador pessoal e uma câmera USB conectada a um

Figura 26 – Arquitetura distribuída do projeto



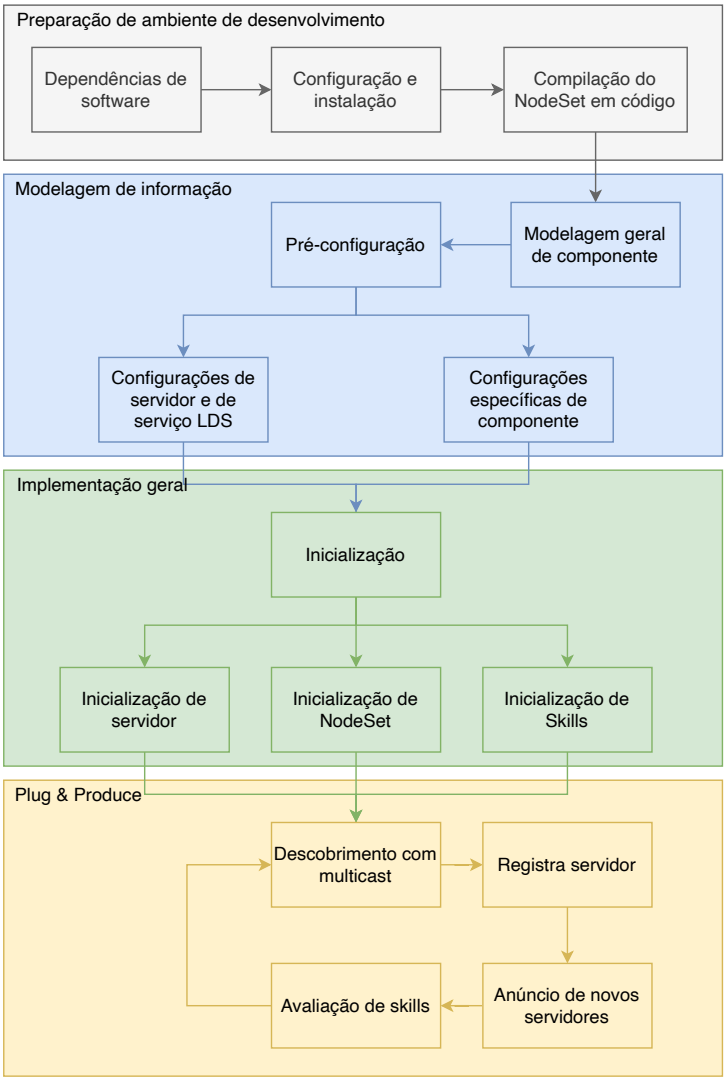
Fonte: autor (2022).

computador embarcado, tal que cada um desses dispositivos executam um servidor OPC UA com a respectiva skill de tirar foto, PhotoSkill. O processador de imagens é implementado em uma máquina virtual em nuvem, do serviço Elastic Compute Cloud (EC2), da AWS. O compositor de skills também é implementado em um computador embarcado, porém que não exige tanto recurso de processamento.

4.2 IMPLEMENTAÇÃO

A Figura 27 mostra os pontos principais do projeto e implementação deste trabalho, divididos nas fases de preparação do ambiente de desenvolvimento, modelagem de informação de componentes, implementação geral de servidores e, finalmente, a implementação da funcionalidade de P&P.

Figura 27 – Visão geral das principais fases de implementação do projeto



Fonte: autor (2022).

4.2.1 Dependências de software

O projeto tem como base principal a biblioteca open62541, introduzida na Seção 2.1.3, escrita na linguagem C99. As fases de desenvolvimento e de depuração requerem alguns softwares a mais. Com exceção de MES, cada componente tem suas dependências específicas, que serão abordadas nas próximas seções.

Na fase de desenvolvimento, as diretrizes de compilação do código são escritas em CMake, utilizando o ambiente de desenvolvimento integrado Virtual Studio Code. Utiliza-se também a ferramenta UA-ModelCompiler, que requer o software Docker[1] instalado. Os servidores são depurados utilizando a ferramenta de cliente OPC UA UaExpert[2], de livre acesso, da empresa Unified Automation GmbH.

4.2.1.1 Dependências gerais e específicas

Para um sistema operacional Linux Ubuntu versão 20.04, há a garantia de que as outras dependências gerais são encontradas no repositório padrão, sendo elas:

- GIT, para download dos repositórios de código.
- Python3 e PIP, com o pacote `netifaces`, para obter os endereços de interfaces de rede em Python.
- mDNS, possibilita a extensão multicast, pois resolve os nomes de host locais em endereços de IP.
- GCC 9.3.0, para compilar o projeto.
- `pkg-config` para consulta de bibliotecas.
- Libconfig, para processar arquivos estruturados de configuração.
- WireGuard, para conexão VPN.

Quanto às dependências específicas, a câmera requer a biblioteca OpenCV para processar e converter imagens. O processador de imagens requer, além da biblioteca OpenCV, a biblioteca ArUco, como descrito na Seção 2.3. O compositor de skills requer a biblioteca Eigen3, para auxiliar nos cálculos matriciais envolvidos na matemática, abordados na Seção 3.

4.2.1.2 Instalação da biblioteca open62541

O recurso de P&P com base em semânticas de skills requer a inclusão de determinados módulos da biblioteca open62541 durante o processo de compilação. Dentre as dependências extras, o módulo de descobrimento com extensão multicast requer a biblioteca `mdns`.

O projeto utiliza a versão 1.3 biblioteca open62541, que deve seguir o processo representado na Listagem 4.4 para um sistema operacional Linux com base em Debian.

[1] https://docs.docker.com/engine/install/
[2] https://www.unified-automation.com/products/development-tools/uaexpert.html

A listagem garante a instalação da biblioteca, tal que possa ser encontrada pela diretriz `find_package`, em CMake.

Listagem 4.4 – Comandos de instalação da biblioteca open62541

```
$ sudo apt install libnss-mdns git build-essential gcc pkg-config
   ↪ cmake python3 python3-pip
$ python3 -m pip install netifaces
$ git clone https://github.com/open62541/open62541.git
$ cd open62541
$ git checkout v1.3
$ git submodule update --init --recursive
$ mkdir build && cd build
$ cmake .. -DUA_ENABLE_SUBSCRIPTIONS_EVENTS=ON -DUA_NAMESPACE_ZERO=
   ↪ FULL -DUA_ENABLE_DISCOVERY=ON -DUA_ENABLE_DISCOVERY_MULTICAST
   ↪ =ON -DBUILD_SHARED_LIBS=ON -DCMAKE_BUILD_TYPE=Release
$ make
$ sudo make install
```

4.2.2 Transformação de NodeSet em código

Seguindo o processo da Figura 9, o arquivo de saída NodeSet2, gerado pela ferramenta UA-ModelCompiler, é, em seguida, repassado à ferramenta NodesetCompiler, integrada à biblioteca open62541, que resultará em um conjunto de código para ser utilizado na implementação. Esse processo é declarado nas instruções de construção do projeto, o arquivo CMakeLists.txt, interpretado pela ferramenta CMake, que utiliza diretrizes de compilação de código importadas da biblioteca open62541.

A Listagem 4.5 mostra o processo para o NodeSet PnPTypes, ao qual é atribuído um índice de Namespace, no caso, 4, e o diretório dos arquivos de saída retornados pela ferramenta UA-ModelCompiler. Ao ser executada, a ferramenta irá retornar os códigos de saída para o diretório de destino especificado por `GENERATE_OUTPUT_DIR`.

Listagem 4.5 – Gerando arquivos de implementação em CMake com a ferramenta NodesetCompiler

```
ua_generate_nodeset_and_datatypes(
  NAME "pnp_types"
  TARGET_PREFIX "${PROJECT_NAME}"
  FILE_CSV "${NODESET_DIR}/PnPTypes/Published/PnPTypes/PnPTypes.csv
     ↪ "
  FILE_BSD "${NODESET_DIR}/PnPTypes/Published/PnPTypes/PnPTypes.
     ↪ Types.bsd"
  OUTPUT_DIR "${GENERATE_OUTPUT_DIR}"
```

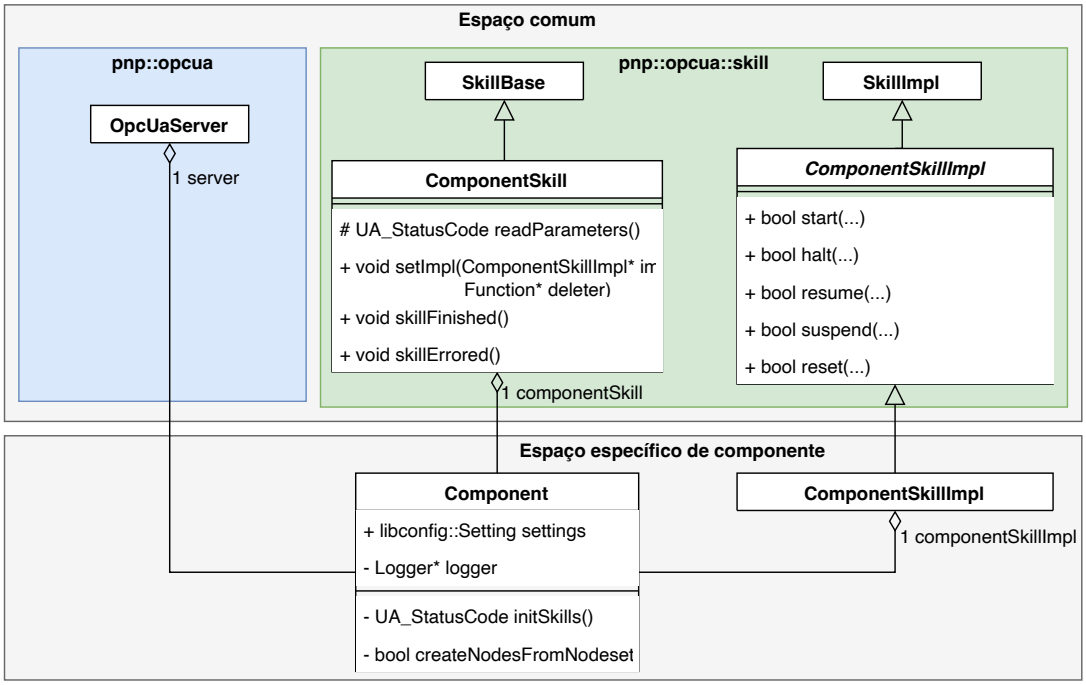
```
  NAMESPACE_MAP "4:https://pnp.org/UA/PnPTypes/"
  FILE_NS "${NODESET_DIR}/PnPTypes/Published/PnPTypes/PnPTypes.
     ↪ NodeSet2.xml"
  DEPENDS "device_model" "di"
  INTERNAL
)
```

A Listagem 4.6 mostra como é definido o tipo de dado PoseDataType, no arquivo resultante `types_pnp_types_generated.h`. Na listagem, define-se primeiro as estruturas para a posição e rotação, que compõem a estrutura de pose, de acordo com a Figura 24.

Listagem 4.6 – Declaração gerada de PoseDataType

```
/**
 * PositionDataType
 * ^^^^^^^^^^^^^^^^
 * A representation of position in cartesian space */
typedef struct {
    UA_Double x;
    UA_Double y;
    UA_Double z;
} UA_PositionDataType;

#define UA_TYPES_PNP_TYPES_POSITIONDATATYPE 0

/**
 * RotationDataType
 * ^^^^^^^^^^^^^^^^
 * A representation of rotation in cartesian space */
typedef struct {
    UA_Double r;
    UA_Double p;
    UA_Double y;
} UA_RotationDataType;

#define UA_TYPES_PNP_TYPES_ROTATIONDATATYPE 1

/**
 * PoseDataType
 * ^^^^^^^^^^^^
 * A representation of pose in cartesian space, composed by
    ↪ position and rotation */
```

Figura 28 – Diagrama de classes de componente genérico



Fonte: autor (2022).

```
29 typedef struct {
30     UA_PositionDataType position;
31     UA_RotationDataType rotation;
32 } UA_PoseDataType;
33
34 #define UA_TYPES_PNP_TYPES_POSEDATATYPE 2
```

Para cada tipo gerado atribui-se um índice definido, no caso do NodeSet PnP Types, de prefixo `UA_TYPES_PNP_TYPES`, utilizado para identificar o tipo de dado na biblioteca open62541. Esses índices correspondem à posição da estrutura do dado em um vetor de tipos, referenciado ao objeto do servidor OPC UA, necessário para os processos de codificação e decodificação de dados.

4.2.3 Modelagem geral de componente

Atribui-se um método de modelagem de código para cada componente, como representado pelo diagrama de classes da Figura 28, supondo um componente genérico qualquer, de nome Component. A princípio, os componentes compartilham de um espaço comum em que se define um servidor OPC UA, que implementa os métodos de descobrimento, classes-base para declaração de skills, e estrutura das skills em si, de cada componente.

A classe ComponentSkillImpl existe tanto no espaço comum de componentes, declarada abstrata, quanto no espaço específico, onde se espera a implementação, de

fato, da skill. Essa mesma classe possui os métodos de transição de estados de skill, conforme as possíveis transições, representadas na Figura 8, pois se salienta que uma skill é derivada de um programa, que, por sua vez, é uma máquina de estados.

A classe ComponentSkill tem a função de especificar a leitura e escrita dos parâmetros da skill, bem como os processos após a troca de estado, representados, na Figura 28, pelos métodos `skillFinished()` e `skillErrored()`. O componente possui uma referência para cada um dos objetos representados e declara um método para inicialização das skills, `initSkills()`, que configura seus parâmetros e atribui a chamada da skill com a sua implementação; possui também um método para criar os nós de seu NodeSet, cujas funções de inicialização são geradas automaticamente pela biblioteca open62541.

4.2.4 Pré-configuração e inicialização de componentes

Apesar de o sistema P&P ser auto-reconfigurável, um processo de pré-configuração é ainda necessário, pois define parâmetros básicos para os dispositivos que serão processados antes da inicialização. Apesar de alguns componentes necessitarem de configurações específicas, como os valores de calibração de uma câmera, alguns dos parâmetros são comuns a todos os dispositivos implementados neste projeto.

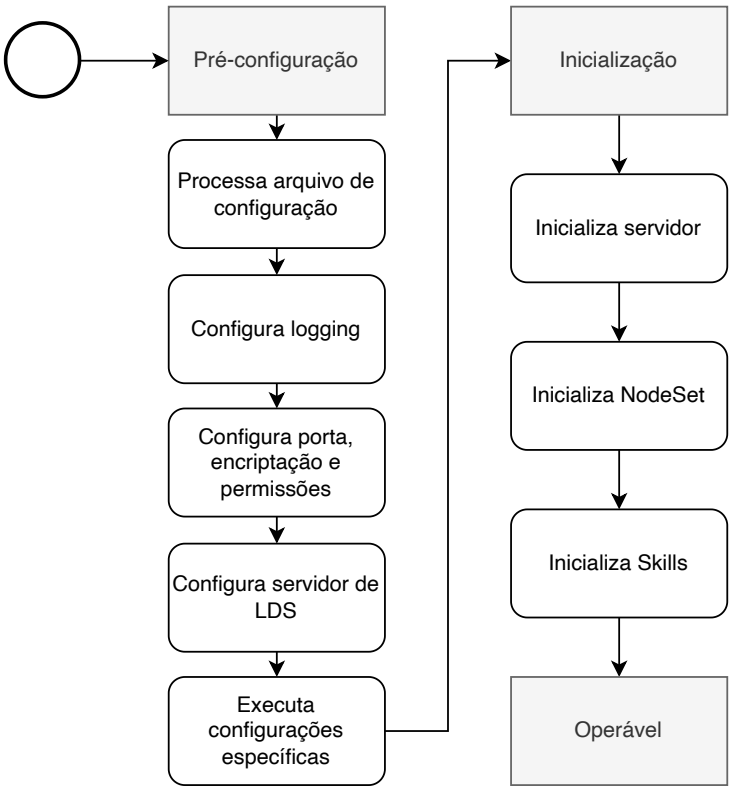
A Figura 29 mostra as etapas de cada um dos processos de pré-configuração e de inicialização de componente, antes de se tornar operável. As configurações são definidas em um arquivo de extensão `.conf`, referenciado no comando de chamada para a execução do servidor, dentro do qual existem, em geral, três grupos importantes de informação: nível de *logging*, parâmetros do servidor (porta de execução, encriptação e permissões de acesso), e endereço do servidor de LDS-ME.

Os parâmetros da pré-configuração são, então, utilizados no processo de inicialização, que, após iniciar o servidor, é responsável por processar os NodeSets gerados a partir da modelagem de informação. Para isso, durante a construção de cada entidade de componente, utilizam-se as funções automaticamente geradas pela biblioteca open62541, que configura e adiciona os nós ao servidor em execução.

O modelo de inicialização de skills segue aquele proposto por Profanter et al. (2021), em que se utiliza de um conjunto de classes abstratas que descrevem cada tipo de skill. Nesse processo, instanciam-se objetos relativos à implementação da skills do componente, ou seja, quais métodos serão chamados após disparar a execução da skill.

Caso um componente falhe durante o processo descrito, de pré-configuração e inicialização, o componente não será operável, portanto, trata-se de uma falha crítica. Quando operável, o dispositivo é autônomo e não requer configurações manuais adicionais.

Figura 29 – Processo geral de pré-configuração e inicialização de componente



Fonte: autor (2022).

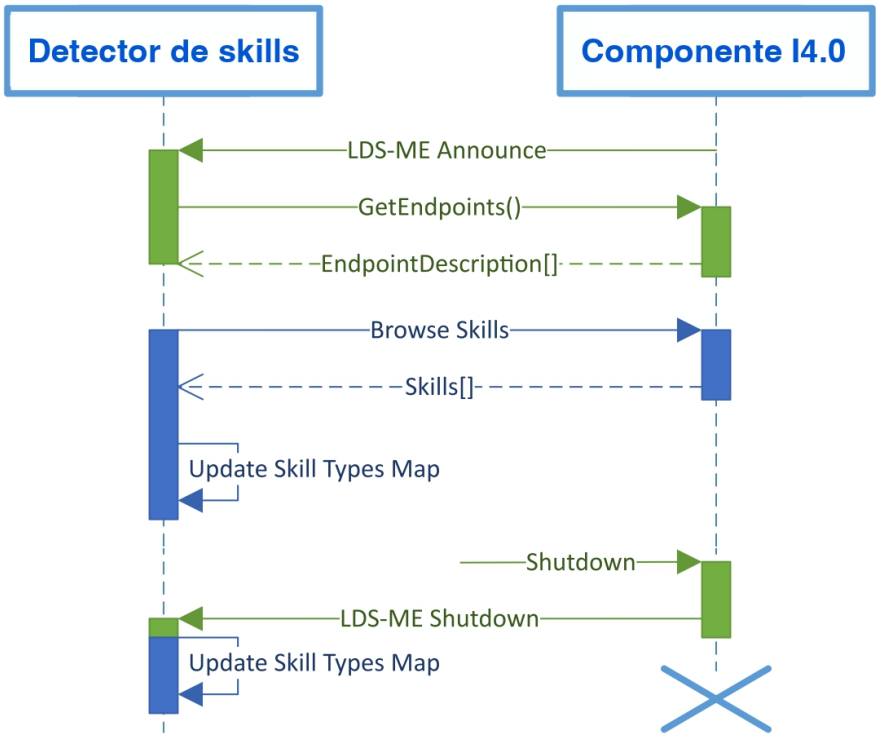
4.2.5 Descobrimento e avaliação de skills

O processo de descobrimento segue o diagrama de sequência da Figura 11, em que um servidor, quando conectado, se registra no servidor de descobrimento, cujo URI é referenciado no arquivo de pré-configuração. O método segue sob gerenciamento da biblioteca open62541 conforme apresentado na Seção 2.2.1, sendo antes necessário configurar o servidor com os parâmetros de descobrimento.

Quando um servidor é descoberto na rede, a biblioteca open62541 dispara um método de chamada de retorno, especificado pelo usuário, porém, gerenciado no projeto dos autores Profanter et al. (2017), que configuram o registro periódico do novo servidor no serviço de LDS-ME. Entretanto, cada componente pode, também, especificar um método de chamada para quando um servidor é descoberto na rede local, possibilitando que ele se conecte com o novo servidor e avalie suas capacidades.

O componente SkillComposer implementa um detector de skills, implementado por Profanter et al. (2021), que realiza o processo de busca de skills para cada novo servidor conectado à rede. A Figura 30 mostra um diagrama de sequência para o processo de descobrimento com busca de skills, em que, ao identificar o Componente I4.0, o detector solicita os dados de conexão com o componente, procura as skills e atualiza seu mapa de skills.

Figura 30 – Processo de detecção de skills



Fonte: adaptado de Profanter (2021, p. 94).

Quando um componente é desconectado da rede, os servidores interessados recebem uma mensagem com o anúncio da desconexão, nisso, o detector de skills atualiza o seu mapa de skills, removendo aquelas do componente desconectado.

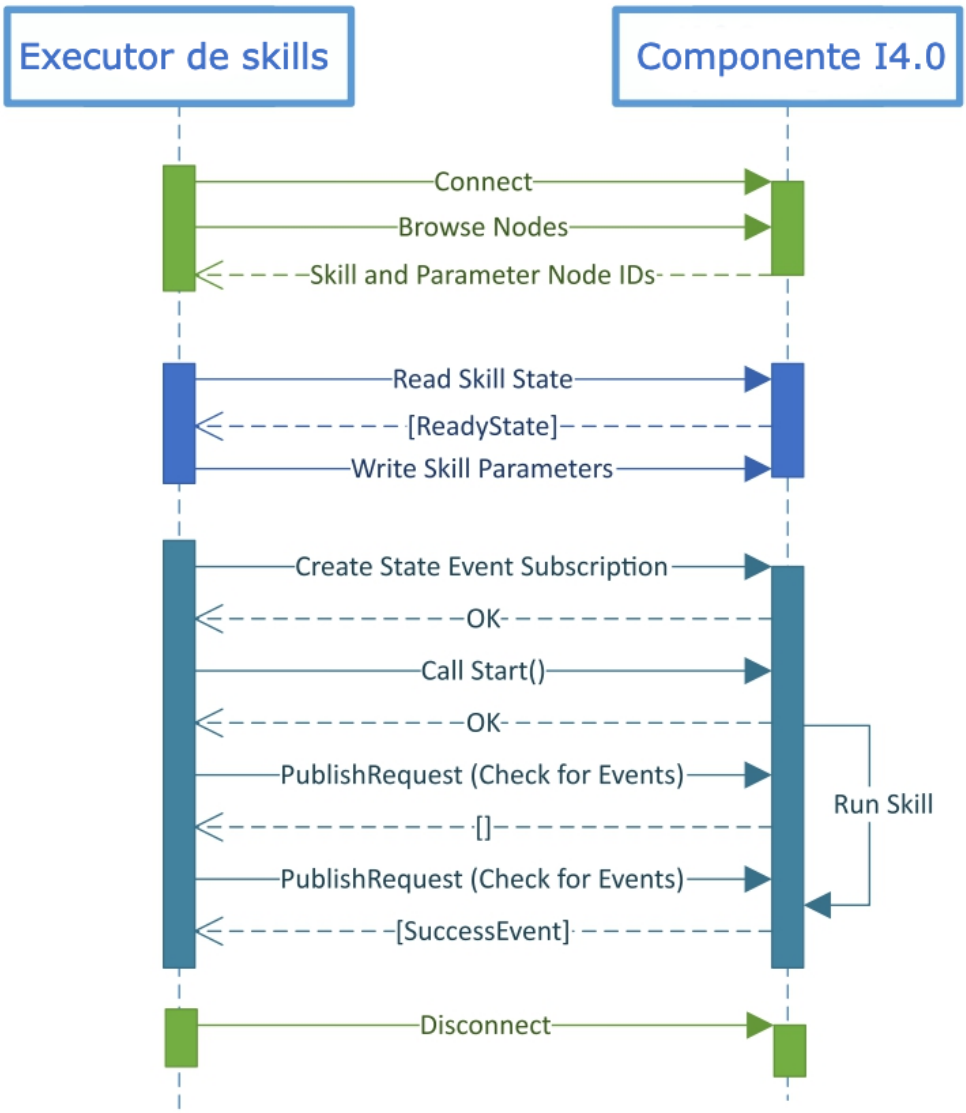
4.2.6 Composição de skills

O compositor realiza o processo de busca de skills utilizando o detector de skills para manter uma lista atualizada das skills disponíveis na rede. Profanter et al. (2021) também implementaram uma classe cliente para execução de skills em outros componentes, cujo diagrama de sequência é mostrado na Figura 31.

Quando chamada MarkerFindingSkill, o compositor de skills verifica se existe algum componente na rede que implementa MarkerDetectionSkill e, caso não existir, alerta uma mensagem de erro e volta para o estado inicial. Em seguida, o compositor de skills procura pelos componentes na rede que implementam PhotoSkill, guardando-os em uma lista, percorrida, câmera por câmera, chamando a execução da skill em todos os componentes.

Ao ler a imagem resultante de todas as câmera, o compositor de skills também lê os valores dos parâmetros das câmeras, isso é, pose e calibração. Os dados são então estruturados e processados no componente de processamento de imagens, que retorna uma lista de quais marcadores fiduciais foram encontrados por câmera.

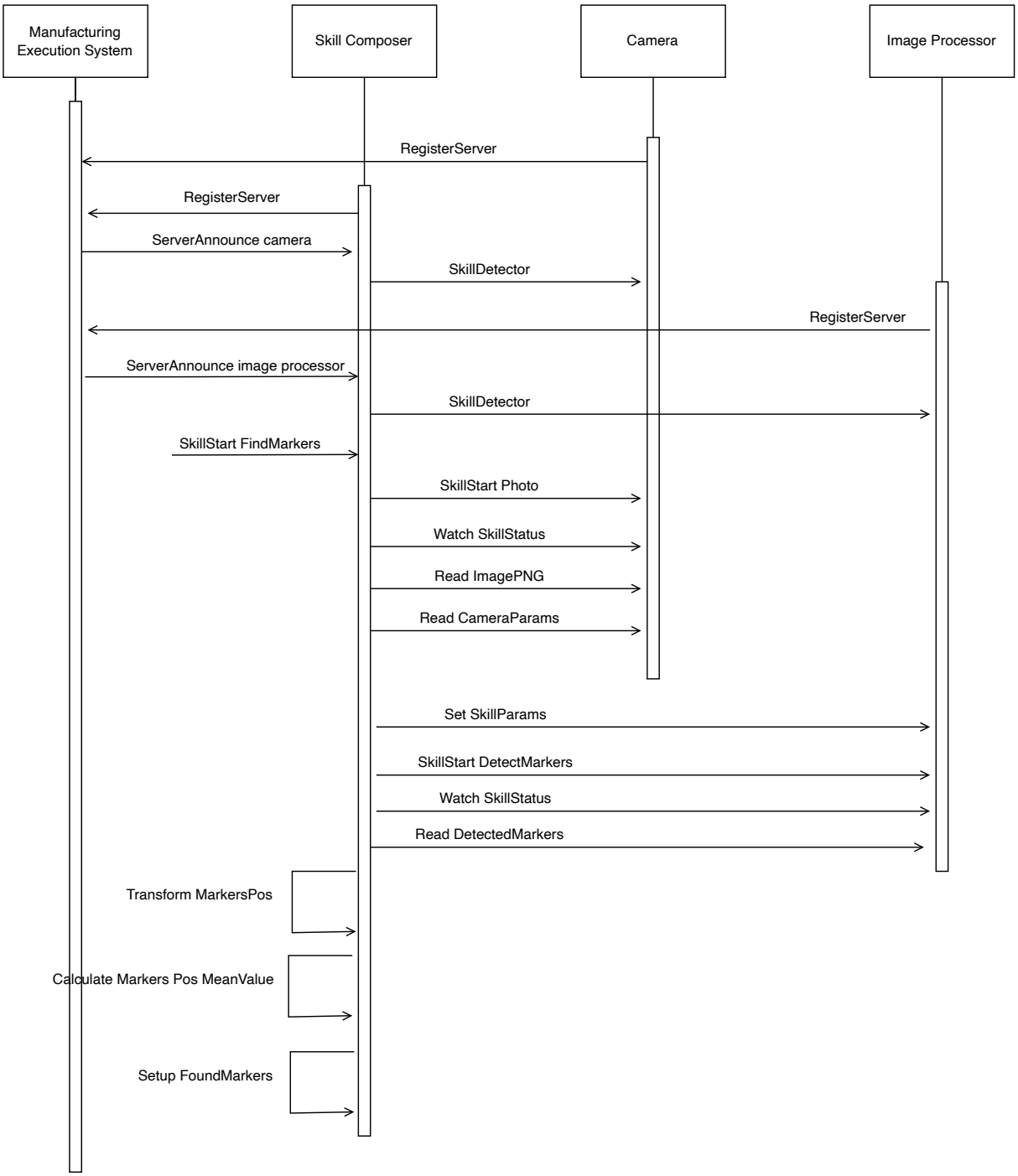
Figura 31 – Processo de execução de skills



Fonte: Profanter (2021, p. 91).

No final do processo, o compositor executa os cálculos de transformação de sistema coordenado para todos os marcadores e os agrupa com base no ID para, em seguida, estabelecer a posição média de cada marcador e, finalmente, escrever os dados na variável de saída. O processo é mostrado no diagrama de sequência da Figura 32, exemplificado em uma rede com apenas uma câmera, mostrando, também, o processo de descobrimento.

Figura 32 – Diagrama de sequência do compositor de skills



Fonte: autor (2022).

5 RESULTADOS E DISCUSSÕES

Implementar o projeto envolveu adaptações dos códigos escritos por Profanter et al. (2021), tanto por conta de alterações nas novas versões da biblioteca open62541, quanto por adaptações ao caso de uso proposto neste trabalho. Como será mostrado nas seguintes seções, a complexidade agregada ao sistema resulta de cerca de quatro mil linhas de código escritas exclusivamente pelo autor deste trabalho.

5.1 ANÁLISE DE SEGMENTOS DE PROGRAMA

A ferramenta `size` do Linux mostra o tamanho das seções em um arquivo objeto, utilizada para analisar as seções de cada um dos componentes implementados. Dessa forma, estima-se, além da memória consumida por cada componente, qual a complexidade envolvida na sua implementação.

A análise é feita com base nos três segmentos de programa: `text`, `data`, `bss`; cujos valores correspondem à quantidade de dados, em bytes. O campo `text` determina a quantidade de código textual do objeto, `data` o tamanho do segmento de dados utilizados e `bss`, *block starting symbol*, o tamanho do segmento de dados não inicializado de cada objeto.

Todos os componentes compartilham os objetos relacionados ao servidor OPC UA com as funcionalidades de descobrimento e gerenciamento de skills, implementados por Profanter et al. (2021). Esses objetos se encontram no diretório `include` e o tamanho de seus segmentos correspondem à primeira linha da Tabela 1, que mostra o tamanho em bytes de cada módulo implementado e os valores totais do sistema.

Tabela 1 – Relação de tamanho de segmentos por módulo do sistema

Texto	Dados	Dados não inicializados	Total	Descrição
2.624.906	20.623	1.553	2.647.082	Bibliotecas comuns, implementam o servidor OPC UA com os recursos de descobrimento e gerenciamento de skills
1.836.386	12.620	2.257	1.851.263	Servidor MES com serviço de descobrimento
2.204.932	15.965	2.569	2.223.466	Componente de câmera
2.055.604	13.908	2.378	2.071.890	Componente de processamento de imagens
2.322.833	15.957	2.569	2.341.359	Componente de composição de skills
11.044.661	79.073	11.326	11.135.060	Total

Fonte: autor (2022).

Observa-se que os elementos implementados por Profanter et al. (2021) correspondem a 23,77% do tamanho total do projeto, tendo a maior participação em quantidade de dados, pois implementa processos complexos de gerenciamento de

skills e da conectividade com os servidores OPC UA. O componente de composição de skills fica em segundo lugar, correspondendo a 21,03% dos dados do sistema, de fato, o processo realizado pelo componente é extenso em termos de quantidade de código, pois é responsável por ler e tratar os dados de câmeras e do processador de imagens.

Apesar de pouca complexidade, a câmera implementa uma interface de código em OpenCV para abstrair os dispositivos de câmeras ou, ainda, imagens e vídeos estáticos, utilizados para simular testes. O processador de imagens utiliza, além da biblioteca OpenCV, a ferramenta ArUco para detectar os marcadores fiduciais, não sendo necessário gerenciar dispositivos. O servidor MES resultou na menor participação no sistema, pois é responsável apenas pelo descobrimento de servidores e não implementa skills.

5.2 COMPONENTES I4.0

Esta seção apresenta os resultados da execução individual dos componentes autônomos, com exceção do servidor MES e do compositor de skills, cuja análise é feita nas Seções 5.3 e 5.4, respectivamente, de testes e experimentos com o sistema. A implementação do sistema foi, primeiro, orientada ao desenvolvimento individual dos componentes de câmera e de processamento de imagens, sendo o restante implementado após garantir a estabilidade funcional desses dois dispositivos.

5.2.1 Câmera

A execução de PhotoSkill utiliza a biblioteca OpenCV para fazer a leitura da imagem de um dispositivo de vídeo do computador ou até de um arquivo de imagem, ou vídeo. Foi definida uma estrutura de configuração específica para a câmera, que determina qual dispositivo de vídeo utilizar, com qual formato de codificação (YUYV, MJPG, etc.), a resolução da câmera, os dados de calibração e pose da câmera.

O teste funcional utilizou uma câmera USB, modelo Logitech C920S, com as pré-configurações definidas conforme a Listagem 5.1. Os valores são automaticamente escritos em seus devidos campos no espaço de endereçamento do servidor OPC UA, o que é conferido utilizando o software UaExpert.

Listagem 5.1 – Configuração específica do teste de câmera USB

```
1 camera = {
2     device = 0
3     width = 2560
4     height = 1472
5     format = "YUYV"
6     camera_info = {
7         distortion_coefficients = {
```

```
            k1 = 0.0
            k2 = 0.0
            p1 = 0.0
            p2 = 0.0
            k3 = 0.0
        }
        camera_matrix = {
            fx = 1.4753717041015625e+03
            cx = 1280.0
            fy = 1.4746522216796875e+03
            cy = 736.0
        }
    }
    camera_pose = {
        position = {
            x = 1.0
            y = 2.0
            z = 3.0
        }
        rotation = {
            r = 0.2
            p = 0.4
            y = 0.8
        }
    }
}
```

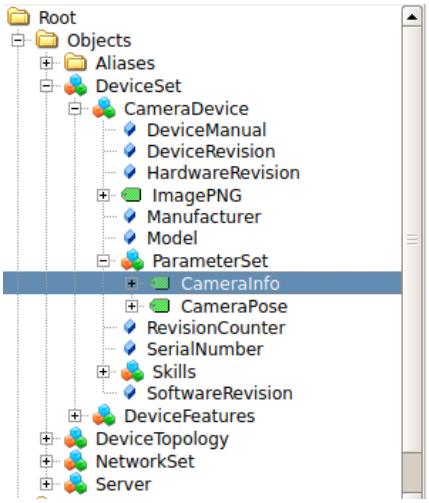
A Figura 33 mostra a visão geral do espaço de endereçamento da câmera, em que se nota o objeto CameraDevice organizado pelo objeto DeviceSet, conforme estabelecido por AAS. A figura também mostra os valores da estrutura CameraInfo, configurados no arquivo de configuração da Listagem 5.1.

A execução de PhotoSkill é mostrada na Figura 34, abordando, também, a visão do espaço de endereçamento do servidor. Utilizando o cliente UaExpert, é possível visualizar a foto tirada pela câmera utilizando o visualizador de imagens, entretanto, nesse teste, o valor da resolução da câmera foi alterado para caber na tela do computador, pois uma alta resolução torna a janela do software muito grande. O teste da regra de negócio RN1 também foi executado e conferiu-se os valores `NAN`, no servidor, configurados para parâmetros indefinidos.

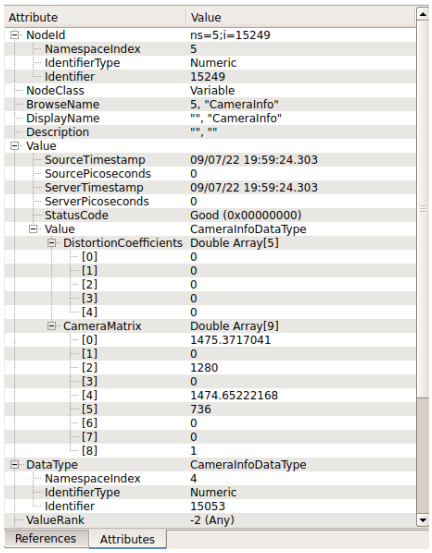
Foram executados outros testes utilizando arquivos de imagem e vídeo, cujo caminho é referenciado no campo `device` do arquivo de configuração. Um computador pode hospedar diferentes câmeras, desde que é provido diferentes arquivos de

Figura 33 – Visão do espaço de endereçamento da câmera

(a) Árvore de nós

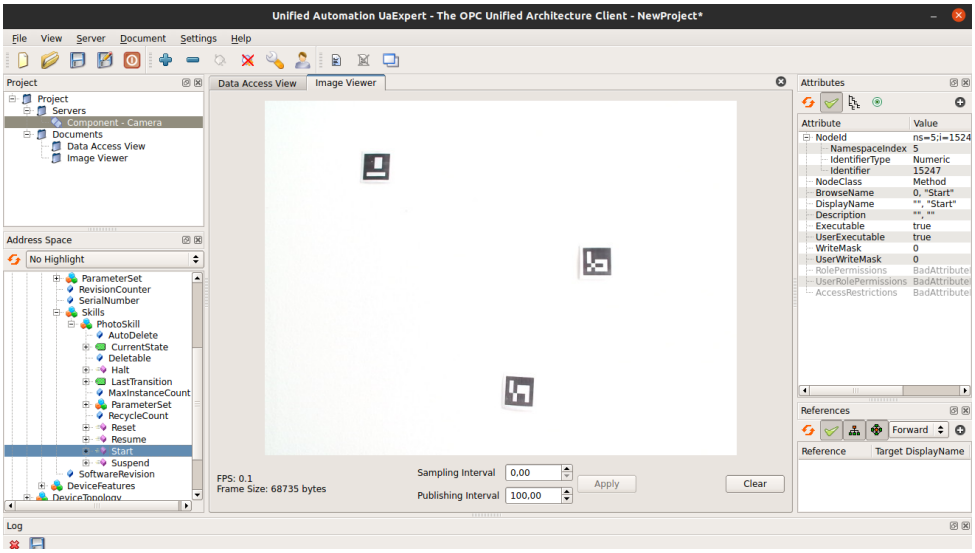


(b) Valores da variável CameraInfo



Fonte: autor (2022).

Figura 34 – Execução de PhotoSkill com visualização de imagem



Fonte: autor (2022).

configuração para cada uma das câmeras.

5.2.2 Processador de imagens

O modelo de configuração para o processador de imagens integra alguns parâmetros específicos da biblioteca ArUco e a especificação de parâmetros de saída. O arquivo de pré-configuração é mostrado na Listagem 5.2, onde o parâmetro `err_ratio` igual a 100 garante a detecção dos marcadores fiduciais mesmo em uma

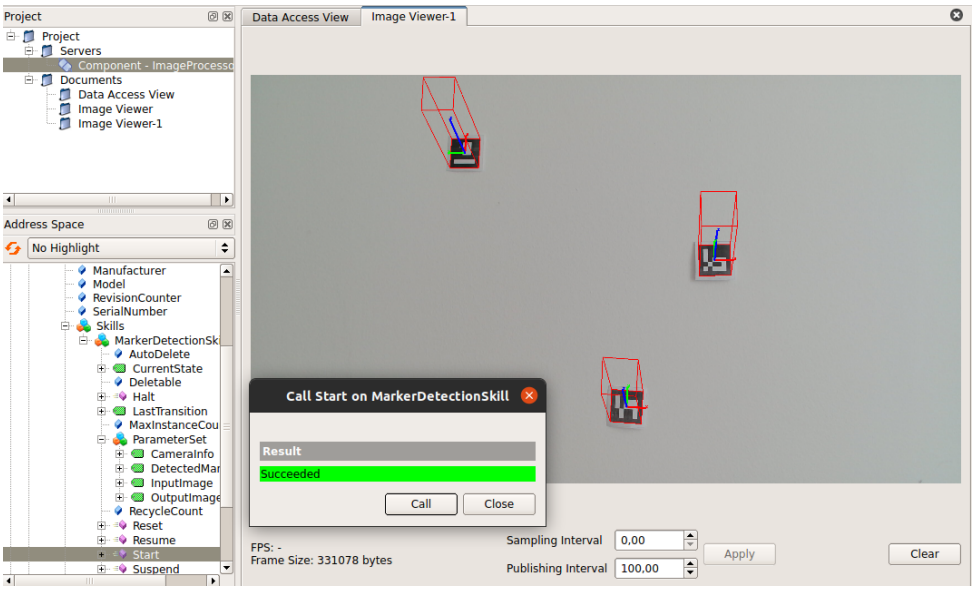
câmera descalibrada. As outras configurações são representativas, ao configurar `load_sample_parameters`, o servidor utiliza dados estáticos de teste como entrada para MarkerDetectionSkill.

Listagem 5.2 – Configuração específica do teste do processador de imagem

```
image_processor = {
    outputImage_resize_height = 300
    err_ratio = 100
    draw_markers = true
    output_image = true
    load_sample_parameters = true
}
```

Os parâmetros `outputImage_resize_height`, `draw_markers` e `output_image` determinam a imagem de saída, que representa visualmente os marcadores detectados. Utilizando dados de teste, a Figura 35 mostra a execução de MarkerDetectionSkill com os parâmetros de teste, onde se visualiza a imagem de saída com a representação gráfica dos marcadores identificados.

Figura 35 – Execução de MarkerDetectionSkill com dados de exemplo

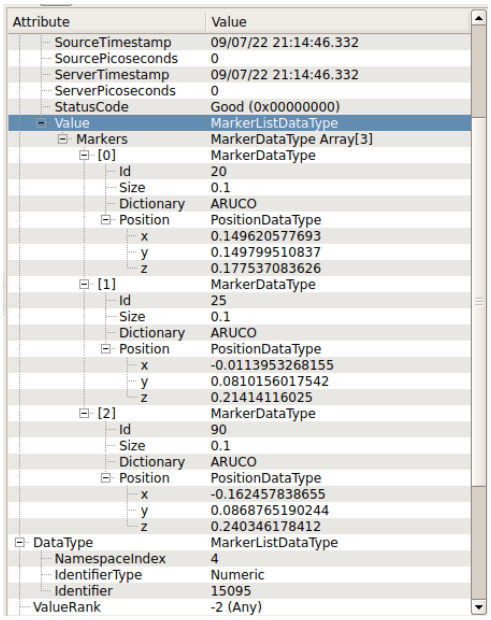


Fonte: autor (2022).

O tamanho dos marcadores são previamente definidos em um arquivo do tipo Comma Separated Values (CSV) e passados como argumento na chamada de execução do servidor. A Figura 36 mostra os marcadores identificados na figura anterior e suas respectivas posições e identificadores.

A lista de marcadores detectados é atualizada a cada execução de MarkerDetectionSkill, mesmo se o número de marcadores for igual a zero, nesse caso, a lista é configurada como vazia. Caso não seja possível detectar a posição de um marcador fiducial, porém apenas o ID, os valores da posição recebem o valor nulo,

Figura 36 – Variável de saída dos marcadores fiduciais detectados



Fonte: autor (2022).

`NAN`, de acordo com a regra de negócio RN1.

5.3 TESTES DE BASE EM MÁQUINA LOCAL

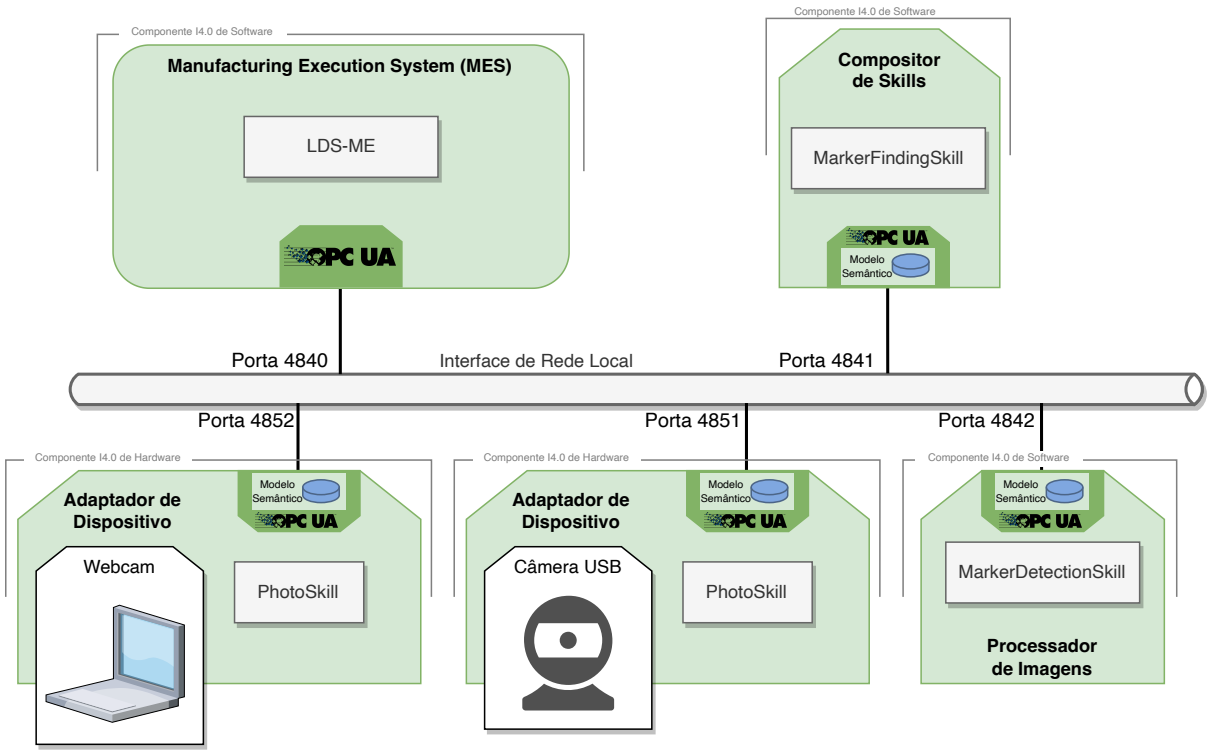
O ambiente principal de testes foi um computador pessoal, com sistema operacional Linux Ubunutu 20.04, processador Intel CORE i5 e 8 GB de memória. O projeto foi compilado e executado nesse computador, distribuindo-se os servidores do sistema em diferentes portas. A Figura 37 mostra a arquitetura base para os testes, com as respectivas portas em que cada servidor está conectado.

5.3.1 Uso de recursos

Foi realizado um teste de execução monitorado por uma ferramenta de análise de uso de memória e CPU, `psutil`, implementada em Python. Primeiro, um *script* executa o servidor MES, após um segundo, executa SkillComposer, após outro segundo, ImageProcessor e, após mais um segundo, executa, juntamente, a Webcam e a câmera USB. Por volta do instante de tempo de 9 segundos, executou-se a procura de marcadores MarkerFindingSkill.

A Tabela 2 mostra o tempo de uso de CPU de cada um dos servidores executados, como também a relação de uso entre eles, ou seja, para a execução do sistema, qual a fração de uso de CPU de cada servidor. O processador de imagens requer o maior tempo de uso da CPU, justificado pelas execuções de detecção de marcadores fiduciais nas duas imagens das câmeras, que ficam em segundo e terceiro

Figura 37 – Arquitetura base de testes



Fonte: autor (2022).

lugar, podendo ser justificado pelo tempo de espera de abertura dos dispositivos.

Tabela 2 – Relação de uso de CPU por servidor

	MES	SkillComposer	ImageProcessor	Webcam	Câmera USB
Tempo de CPU (s)	0,52	0,39	0,92	0,75	0,72
Relação	15,76%	11,82%	27,88%	22,73%	21,82%

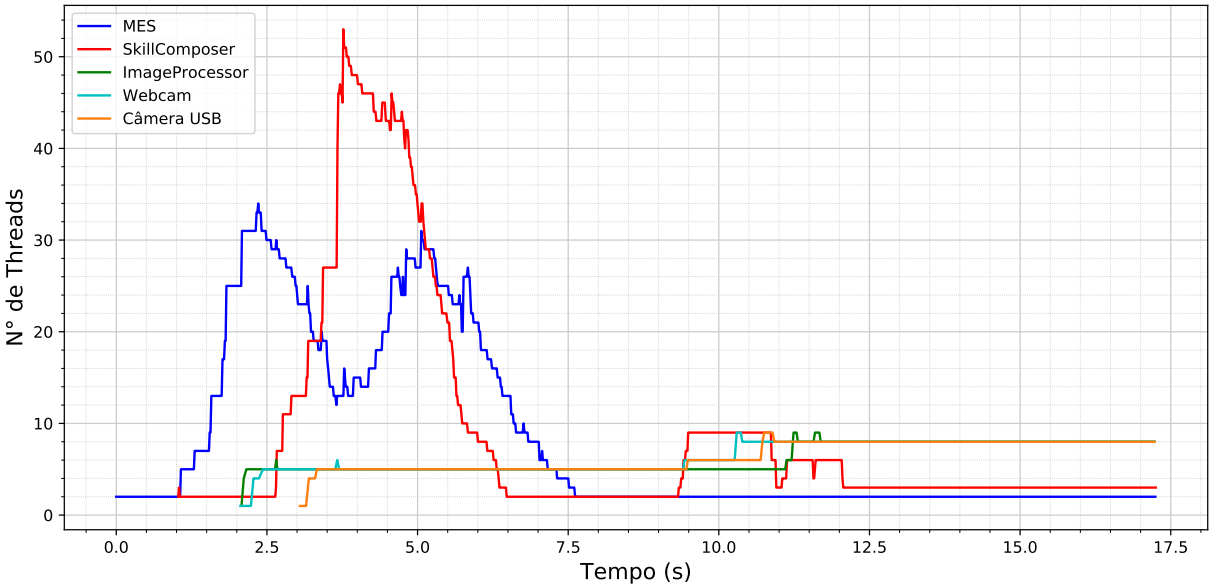
Fonte: autor (2022).

Entretanto, o compositor de skills e o servidor MES criam consideravelmente mais *threads* do que os demais servidores, apontado na Figura 38. Observa-se que o acúmulo de *threads* ocorre nos instantes em que outros servidores são iniciados.

De fato, MES e SkillComposer são os únicos servidores que processam as informações de descobrimento dos demais, que ocorre por meio do estabelecimento de um canal de comunicação com o servidor descoberto. Ainda, o compositor de skills avalia as skills de cada servidor conectado, conforme mostrado no diagrama de sequência da Figura 30, fazendo com que a biblioteca open62541 e os códigos escritos por Profanter et al. (2021) chegam a operar com 53 *threads* em um único instante.

Nota-se, também, que mesmo após o processo de procura de marcadores, encerrado por volta do instante de tempo de 11,5 segundos, os servidores permanecem com mais *threads* ativas do que quando ociosos, indicando um possível ponto de comportamento não-ótimo.

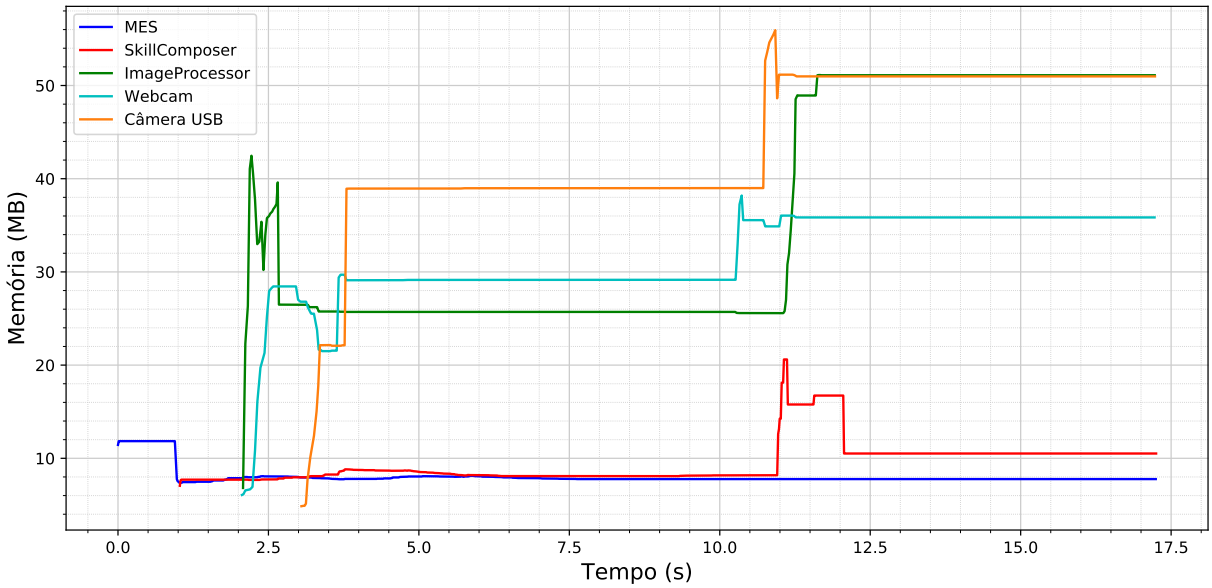
Figura 38 – Quantidade de threads utilizadas por cada servidor



Fonte: autor (2022).

A Figura 39 mostra a quantidade de memória alocada por cada servidor. O consumo de memória das câmeras é justificado pela transmissão de imagem, pois os dispositivos são abertos e inicializados com a execução do servidor. Entretanto, ImageProcessor também inicializa com um pico de memória alocada, pois, durante o teste o servidor estava configurado para carregar uma imagem de teste para a depuração da detecção de marcadores. Por outro lado, observa-se que o processo de descobrimento não sofre picos consideráveis de alocação de memória.

Figura 39 – Memória alocada por servidor



Fonte: autor (2022).

O gráfico de uso de memória permite identificar o processo de procura de

marcadores fiduciais. Sendo o tempo de 9 segundos o instante marcado da execução de MarkerFindingSkill, nota-se que a primeira câmera gravou a imagem no instante de 10,25 segundos e, a segunda câmera, por volta de 0,5 segundos depois, no instante de 10,75 segundos. Logo, o processo de tirar foto nas duas câmeras sofre um atraso de 1,75 segundos.

Para que o sistema seja sincronizável, deve-se considerar os intervalos de tempo entre as fotos das câmeras, o que pode ser resolvido por meio de uma variável de *timestamp* na estrutura de dados da foto da câmera. Essa correção pode ser introduzida na modelagem de informação dos dispositivos de câmera, declarando um parâmetro de saída, ou, utilizando uma relação de propriedade para o objeto ImagePNG.

Após o processo da última câmera, o compositor de skills lê e repassa foto por foto para o processador de imagens, que sofre um pico de memória e, depois, outro pequeno pico, em um intervalo de aproximadamente 0,5 segundos, estimado o tempo de processamento por imagem. Ao finalizar o processo de procura de marcadores, ImageProcessor segue com praticamente o mesmo tanto de memória alocada que a câmera USB, processada por último. SkillComposer não retorna para a quantidade de memória inicial, pois, após o processo, escreve os dados de saída em seu espaço de endereçamento.

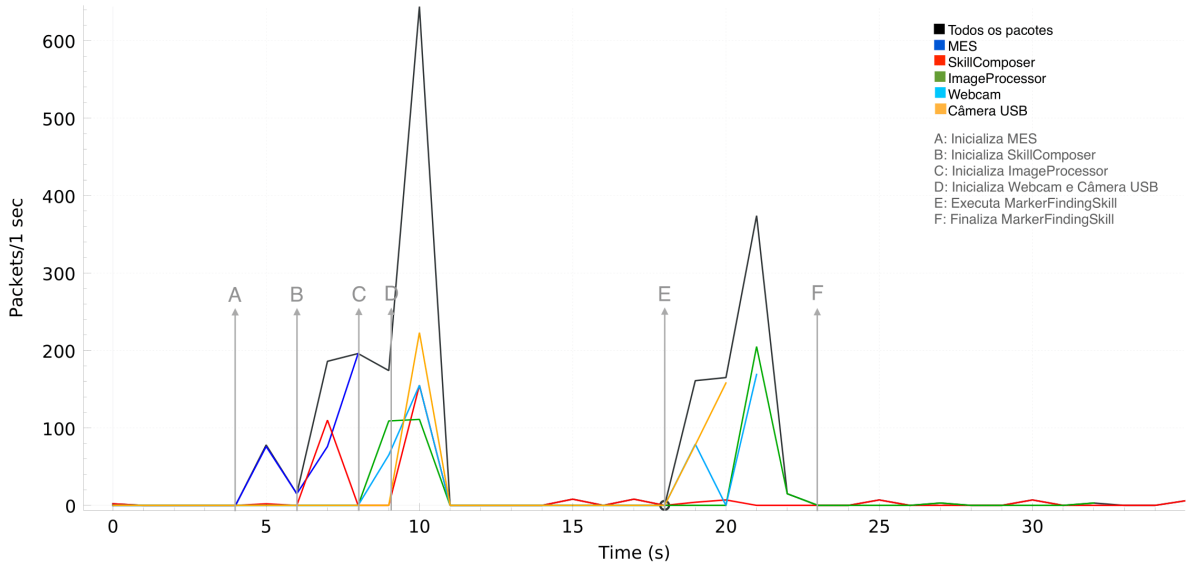
5.3.2 Uso de rede

O software Wireshark foi utilizado para fazer a análise do tráfego de rede resultante do teste com a arquitetura da Figura 37. O fluxo de execução é descrito com o auxílio da Figura 40, que mostra os respectivos instantes que cada servidor é inicializado, o fluxo gerado, de entrada e saída, e a chamada de MarkerFindingSkill, no componente de composição de skills, SkillComposer.

Nota-se que cada servidor gera um fluxo de 80 a 220 pacotes por segundo quando inicializados, que aumenta gradualmente em função do número de servidores já inicializados no sistema, resultante do processo de descobrimento, pois, cada servidor recebe uma lista dos componentes registrados na rede (Figura 11). Através do gráfico da Figura 40, estima-se que o incremento do fluxo de pacotes a cada servidor iniciado é de cerca de 40 pacotes por segundo.

Os instantes referenciados por E e F, na Figura 40, marcam o processo de procura de marcadores nas duas câmeras do sistema. No mesmo instante E, o componente SkillComposer executa PhotoSkill nas duas câmeras. A câmera USB é processada primeiro, ao terminar a execução de PhotoSkill, e gera um tráfego de cerca de 160 pacotes por segundo. Em seguida, a Webcam é processada e gera ligeiramente mais pacotes que a câmera USB. A ascensão da curva verde caracteriza o processo de detecção de marcadores fiduciais, utilizando o processador de imagens, ImageProcessor, com fluxo de pouco mais de 200 pacotes por segundo.

Figura 40 – Fluxo de pacotes de entrada e saída do sistema



Fonte: autor (2022).

A Tabela 3 mostra a quantidade de pacotes por protocolo de camada de rede, especificando, também, a quantidade total de dados atribuída a cada protocolo. Nota-se que a participação de dados específicos do padrão OPC UA corresponde a apenas 0,41% do total transferido, sendo a maior parte atribuída aos pacotes Transmission Control Protocol (TCP) trocados entre os servidores.

Tabela 3 – Transferência de dados por protocolo de camada de rede

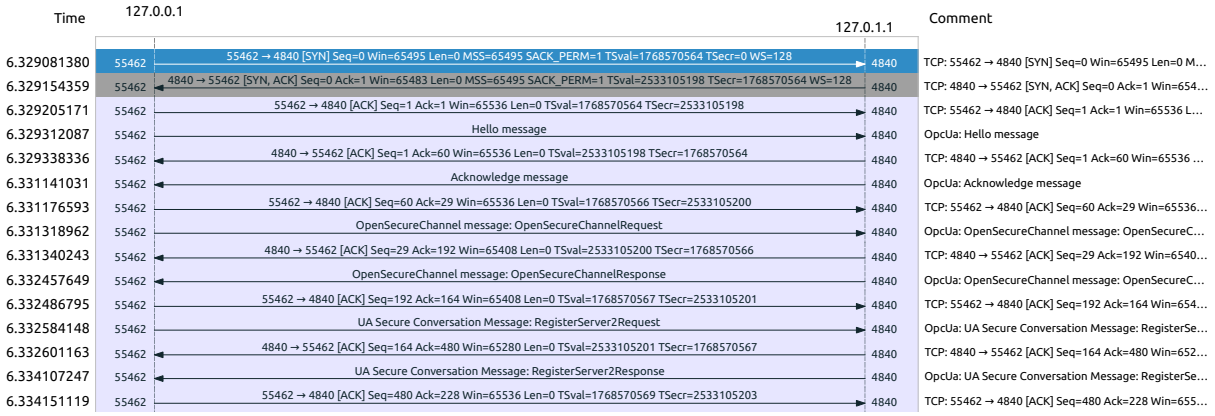
Protocolo	% Pacotes	n° Pacotes	% Bytes	Bytes
Frame	100,00	2052	100,00	4.787.247
Ethernet	100,00	2052	0,60	28.728
IPv4	100,00	2052	0,86	41.040
TCP	100,00	2052	98,54	4.717.479
OPC UA	6,82	140	0,41	19.615
Dados	43,23	887	96,75	4.631.552

Fonte: autor (2022).

De fato, dois servidores OPC UA utilizam a comunicação TCP comum para trocarem informações. Os dados da camada de rede IPv4 e da camada de dados Ethernet estão relacionados com os cabeçalhos das mensagens e, juntas, correspondem a 1,46% de todos os dados transferidos.

Ainda utilizando o gráfico da Figura 40, mapeia-se, a partir do tempo, a troca de mensagens do processo de descobrimento entre o servidor MES e o componente de composição de skills. A Figura 41 mostra o diagrama de sequência representando o fluxo de mensagens entre os dois servidores, já após se comunicarem pelo processo de multicast.

Figura 41 – Fluxo de mensagens do processo de descobrimento



Fonte: autor (2022).

As mensagens do padrão OPC UA são criptografadas, portanto, não são exibidas no Wireshark, porém, ainda é possível observar informações relevantes do padrão, como a mensagem de *hello*, na Figura 40, respondida com um *ACK*, provavelmente contendo as informações de conexão com o servidor MES, pois, em seguida, estabelece-se um canal seguro de comunicação. As duas ultimas mensagens OPC UA caracterizam o descobrimento na conversação, pois mostram a requisição do serviço RegisterServer2, utilizado para registrar os endereços de descobrimento do servidor. Essa informação é reconhecida pelo Wireshark e a mensagem OPC UA de RegisterServer2Request é mostrada na Listagem 5.3.

Listagem 5.3 – Mensagem de registro de servidor no processo de descobrimento em OPC UA

```
OpcUa Binary Protocol
    Message Type: MSG
    Chunk Type: F
    Message Size: 288
    SecureChannelId: 4
    Security Token Id: 4
    Security Sequence Number: 2
    Security RequestId: 2
    OpcUa Service : Encodeable Object
        TypeId : ExpandedNodeId
        RegisterServer2Request
            RequestHeader: RequestHeader
            Server: RegisteredServer
                ServerUri: pnp.component.skillcomposer
                ProductUri: http://open62541.org
                ServerNames: Array of LocalizedText
                ApplicationType: Server (0x00000000)
```

```
18                  GatewayServerUri: [OpcUa Null String]
19                  DiscoveryUrls: Array of String
20                      ArraySize: 2
21                      [0]: DiscoveryUrls: opc.tcp://lucas-laptop
                            ↪ :4841/
22                      [1]: DiscoveryUrls: opc.tcp://lucas-laptop
                            ↪ :4841/
23                  SemaphoreFilePath: [OpcUa Null String]
24                  IsOnline: True
25              DiscoveryConfiguration: Array of ExtensionObject
26                  ArraySize: 1
27                  [0]: ExtensionObject
28                      TypeId: ExpandedNodeId
29                      EncodingMask: 0x01, has binary body
30                      MdnsDiscoveryConfiguration:
                            ↪ MdnsDiscoveryConfiguration
```

O componente SkillComposer registra-se no servidor MES dizendo dois endereços de descobrimento, cuja razão é a existência de um segundo endereço de *loopback*, 127.0.1.1, gerado pelo serviço de mDNS, conforme observado na Figura 41, em que no caso, o compositor de skills utiliza um cliente com um *socket*, de valor 55462, na interface de *loopback* padrão, 127.0.0.1, para se comunicar com o servidor descoberto na interface de mDNS. No final da Listagem 5.3 encontra-se a especificação da configuração de descobrimento por mDNS, que traduz o URI do servidor, no caso, `pnp.component.skillcomposer`, no endereço de IP necessário para conexão, nesse caso, os dois endereços de *loopback*.

5.4 EXPERIMENTOS EM ARQUITETURA DISTRIBUÍDA COM VPN

Esta seção apresenta os resultados dos experimentos utilizando a arquitetura distribuída em uma rede VPN, proposta na Seção 4.1, representada na Figura 26. Utilizando a ferramente WireGuard, foi configurada uma sub-rede virtual 10.0.0.0/24 e, a cada componente, foi atribuído um endereço de IP.

Primeiramente, foi tentado garantir o funcionamento do descobrimento através de Domain Name System (DNS) com multicast na rede virtual VPN. Dessa forma, o servidor MES foi executado na máquina virtual em nuvem, EC2, com a porta 51820 exposta à Internet, atribuindo-a o IP privado 10.0.0.1. Do outro lado, ao laptop com a webcam foi atribuído o IP privado 10.0.0.2 e, como garantia, todo tráfego foi redirecionado à máquina virtual EC2.

Com essa configuração, as mensagens de multicast geradas pelo laptop não atravessaram a sub-rede, portanto, não foram enviadas à máquina virtual pela interface

Figura 42 – Fluxo de mensagens de mDNS entre dois servidores em uma rede VPN

Fonte: autor (2022).

do WireGuard. Logo, foram realizadas algumas alterações, no caso, a configuração da interface do WireGuard com multicast ativado e o uso da ferramenta SMCRoute, que executa o roteamento estático de pacotes de multicast.

Utilizando o Wireshark, observou-se que a ferramenta mDNS utiliza o endereço de multicast 224.0.0.251, configurado para ser roteado para a interface do WireGuard nos dois computadores. Nessa segunda tentativa, as mensagens de multicast atravessaram a sub-rede e, conforme mostra o diagrama de fluxo de mensagens da Figura 42, o servidor MES, da máquina virtual, enviou respostas de multicast de volta para o laptop.

Observa-se, na Figura 42, que o laptop envia, na mensagem de multicast, o nome do servidor OPC UA conectado, `pnp.component.camera`, seguido do URI do servidor, `lucas-laptop`, especificando também a porta de conexão 4848. Da mesma forma, o servidor MES retorna a mensagem, porém, agora, diretamente ao laptop, com o nome do servidor seguido do URI e mais a porta padrão de conexão 4840.

O processo de descobrimento proposto por Profanter et al. (2017) utiliza o serviço de DNS para resolver o endereço URI e obter o IP do servidor da ponta, ou seja, o laptop tenta obter o endereço de conexão com a máquina virtual após descobrir o seu URI. Entretanto, esse processo falha, pois o laptop não consegue resolver o endereço, como mostram as mensagens de saída da execução do servidor, na Listagem 5.4.

Listagem 5.4 – Tentativa falha de resolução de endereço utilizando DNS com VPN

```
[info     ] [camera] OpcUaServer::onServerAnnounce
[warning  ] [camera-ua-client] [OPC UA/userland] AcceptAll
   ↪ Certificate Verification. Any remote certificate will be
   ↪ accepted.
[warning  ] [camera-ua-client] [OPC UA/network] DNS lookup of ip
   ↪ -172-31-41-63 failed with error -3 - Unknown error
[warning  ] [camera-ua-client] [OPC UA/client] Could not open a TCP
   ↪ connection to opc.tcp://ip-172-31-41-63:4840
[error    ] [camera] Could not read StartTime from LDS:
   ↪ BadServerNotConnected}]
```

Da mesma forma, a máquina virtual não consegue se conectar com o laptop, acusando o erro de que não possível encontrar *endpoints* para o endereço de descobrimento `opc.tcp://lucas-laptop:4848/`. Portanto, não foi possível executar o sistema proposto em uma rede distribuída, conectada com VPN. Uma das possíveis justificativas, é que os endereços de multicast entre 224.0.0.0 e 224.0.0.255 são reservados para protocolos de rede no mesmo segmento local (CHESHIRE; KROCHMAL, 2013).

5.5 EXPERIMENTOS EM REDE LOCAL

Uma arquitetura semelhante à da Figura 26 foi utilizada para o teste do sistema distribuído localmente, colocando o processador de imagens em um computador pessoal, ao invés de utilizar uma nuvem e, descartando o uso de VPN. O servidor MES foi também executado no computador pessoal, o mesmo do componente Webcam, restando dois Raspberry Pi como dispositivos de ponta, sendo o Raspberry Pi 4 o servidor da câmera USB e o Raspberry Pi Z o servidor do compositor de skills.

Como preparação, foi garantido que as portas dos servidores em seus devidos dispositivos são expostas à rede local para permitir o acesso mútuo entre componentes. Entretanto, ocorreu que os dois Raspberry Pi possuíam, por padrão, o mesmo nome de *host* configurado: `raspberrypi`. Apesar de alterados os nomes para `rpi4` e `rpiz`, foi necessário modificar o arquivo de hosts do Linux, de caminho `/etc/hosts`, para atribuir os nomes à interface local de IP 127.0.1.1, por outro lado, ocorria conflito de resolução de URI na rede.

Ao iniciar o processo de procura de marcadores no sistema, o compositor de skills executou com sucesso até a etapa de leitura de imagem em uma das câmeras, que retornou erro de conexão. Investigando a causa do erro, utilizando o software Wireshark, foram observadas várias retransmissões de pacotes TCP e conexões quebradas, ocorrendo cerca de 10 conexões entre o computador e o Raspberry Pi a cada teste. A

Figura 43 – Retransmissões na abertura de canal de comunicação entre dois dispositivos na rede local

Fonte: autor (2022).

Figura 43 mostra uma tentativa de abertura de canal seguro de comunicação entre os dois dispositivos.

As retransmissões caracterizam pacotes perdidos, o erro *TCP Dup ACK* sinaliza possíveis pacotes sendo entregues fora de ordem. *Spurious Retransmission* são pacotes retransmitidos sem necessidade. O último erro, em vermelho, aponta a quebra da conexão, nesse caso, por conta da finalização do programa.

Após PhotoSkill ser executada, o compositor de skills deveria se reconectar ao dispositivo de câmera, caso não há conexão, e efetuar a leitura da imagem. Entretanto, a tentativa de reconexão, apesar de apontada na saída do programa, segundo os dados de captura do Wireshark, não cria nenhuma solicitação de conexão com o Raspberry Pi. Logo, apesar dos dispositivos funcionarem corretamente de forma individual utilizando o software cliente UaExpert, o compositor de skills falha ao ser executado em uma rede local com dispositivos distribuídos.

5.6 PREENCHIMENTO DE REQUISITOS E OBJETIVOS

Esta seção revisa e discute os resultados dos objetivos e requisitos propostos, respectivamente, nas Seções 1.2 e 3.3, conforme os quadros apresentados a seguir.

O Quadro 1 mostra a relação de preenchimento dos objetivos específicos, apenas não cumprido aquele que se refere ao funcionamento das funcionalidades de P&P do sistema em redes distribuídas. O objetivo geral, descrito na Seção 1.1, foi

concluído, pois, o sistema implementado se baseou em técnicas e métodos levantadas na fundamentação teórica, Capítulo 2, e esquematizado conforme os diagramas ao longo da metodologia, no Capítulo 4, utilizando a biblioteca de código aberto open62541 e Raspberry Pis como dispositivos embarcados.

No Quadro 2, mostra-se a relação de preenchimento de requisitos funcionais, não funcionais e de regras de negócio, comentando cada um dos casos. De forma geral, a análise mostra que o preenchimento de requisitos foi de 71,43%, em que três das falhas estão atribuídas ao não funcionamento do sistema em arquiteturas distribuídas.

Quadro 1 – Resultados de preenchimento de objetivos específicos

Objetivo específico	*Preenchido*	*Comentário*
Introduzir os princípios básicos de modelagem de informação no padrão de comunicação OPC UA	Sim	O espaço de endereçamento e modelagem básica de informação em OPC UA são apresentados na Seção 2.1
Estudar as normas e técnicas para modelagem de componentes com capacidade P&P	Sim	Estudos prévios sobre componentes P&P fundamentam a metologia aplicada no trabalho, na Seção 2.2
Conhecer as ferramentas para implementação no padrão OPC UA dos componentes modelados	Sim	Ferramentas de construção de NodeSets e a biblioteca base utilizada são apresentadas no Capítulo 4
Esquematizar o processo de descobrimento e interconexão de dispositivos OPC UA	Sim	O processo de descobrimento é descrito na Seção 2.2.1 e mostrado, em prática, na Seção 5.3.2
Aplicar a biblioteca de código livre open62541 para implementar uma rede de componentes com a funcionalidade de detecção de posição de marcadores fiduciais, composta por câmeras e um processador de imagens	Sim	A detecção automatizada de marcadores fiduciais foi implementada com sucesso em uma máquina local conforme a descrição na Seção 5.3
Demonstrar a funcionalidade de P&P tanto por meio de um sistema conectado por uma rede local, quanto por uma rede virtual distribuída	Não	Não se obteve sucesso ao mostrar as funcionalidades de P&P tanto em uma rede virtual quanto em uma rede local, respectivamente, conforme as Seções 5.4 e 5.5
Avaliar as características positivas e negativas da metodologia proposta com base em um conjunto de requisitos	Sim	As características positivas e negativas da abordagem utilizada são apresentadas como conclusão, no Capítulo 6

Fonte: autor (2022).

Quadro 2 – Resultados de preenchimento de requisitos

Req.	*Descrição*	*Preenchido*	*Comentário*
RF1	O descobrimento é gerenciado pelo OPC UA e acontece logo após a conexão de dispositivos à rede	Sim	Utilizando o Wireshark, foram mostradas as mensagens de multicast referentes ao descobrimento logo após a inicialização dos servidores
RF2	Os componentes podem ser configurados para serem avisados quando outros dispositivos são conectados	Sim	Aplicado para o compositor de skills, que realiza a avaliação de skills do novo dispositivo logo após seu descobrimento
RF3	Os dispositivos ficam disponíveis imediatamente após a conexão	Sim	A conexão à rede é estabelecida durante o processo de inicialização, que pré configura o dispositivo e o torna operável
RF4	A desconexão de um dispositivo não trava a execução do sistema	Sim	Não foram observados casos em que o sistema sofreu falhas devido a dispositivos desconectados
RF5	As exceções disparadas ao longo da execução são tratadas pelo sistema	Não	O compositor de skills não trata o erro de conexão quando em um sistema distribuído, resultando em falha de execução
RF6	Todo dado é válido e a falta de pré-configuração de um sistema não implica que o modelo é não-operável, mas se comportará dentro de suas capacidades	Sim	Valores não configurados recebem um valor padrão ou o valor inválido `NAN`
RF7	O sistema pode ser executado de forma distribuída	Não	Os experimentos de arquitetura distribuída tanto em rede local quanto em rede virtual não obtiveram sucesso
RNF1	A autonomia é garantida pela acessibilidade à infraestrutura digital	Sim	Os servidores são autônomos e acessíveis em todas as arquiteturas propostas
RNF2	A interoperabilidade é garantida pelos padrões de semânticas	Sim	O modelo semântico garantiu acesso às funcionalidades independente da natureza do sistema, por exemplo o sucesso utilizando câmeras webcam integradas ou USB
RNF3	O sistema é descentralizado e os recursos são utilizáveis mesmo quando separados	Não	A utilização dos recursos de composição de skill requer uma conexão em rede local
RNF4	O sistema pode ser composto por diferentes modelos de câmera	Sim	O requisito é possibilitado pelo uso da biblioteca OpenCV para gerenciar as câmeras, testado com webcams integradas e câmeras USB
RNF5	O sistema é de plataforma independente	Não	Só se conseguiu implementar o sistema em sistemas operacionais Linux de distribuição baseada em Debian. Não se obteve sucesso ao compilar o sistema em MacOS e a plataforma Windows não foi testada
RN1	Uma variável numérica não configurada recebe o valor `NAN`	Sim	Implementado em código e observado nos resultados
RN2	Se disponível, um marcador sempre herda a posição válida	Sim	Comportamento observado de acordo com o esperado

Fonte: autor (2022).

6 CONCLUSÃO

O processo da Quarta Revolução Industrial é guiado e documentado pela instituição governamental alemã Plattform Industrie 4.0, que disponibiliza conjuntos de normas e modelos de referência para o desenvolvimento de ecossistemas digitais baseados nos pilares da interoperabilidade, autonomia e sustentabilidade. Dentro desse campo de pesquisa, foi discutido o caso de uso das fábricas adaptáveis e como que o modelo de P&P possibilita reduzir a necessidade dos processos de reconfiguração nas indústrias ao efetuar modificações nos sistemas de produção.

Recomendado pelo modelo de referência RAMI4.0, o padrão de comunicação industrial OPC UA foi utilizado como base de modelagem e implementação de um sistema de arquitetura orientada a serviços, que, de forma abstrata, descreve componentes de produção em função de suas funcionalidades e recursos. Isso se fez pelo mapeamento da camada de abstração AAS no espaço de endereçamento do OPC UA, mostrando os conceitos fundamentais de modelagem de informação no padrão.

O projeto foi baseado em um trabalho anterior, realizado por Profanter et al. (2021), em que, utilizando uma série de artigos anteriores como referência, foi implementado em OPC UA um sistema funcional de P&P com base no uso de semânticas de skills, uma forma de abstrair as funções de dispositivos de produção em um modelo executável de máquinas de estados finitos. O código implementado pelos autores corresponde a cerca de 24% da carga de dados do resultado deste trabalho, que provê uma biblioteca para o servidor OPC UA com os recursos de descobrimento e gerenciamento de skills.

Utilizando um demonstrador base, composto por câmeras e um processador de imagem que efetua a procura automática de marcadores fiduciais, este trabalho mostrou como que as informações do projeto podem ser modeladas em arquivos de texto XML e convertidas para o espaço de endereçamento de cada servidor OPC UA. A automatização desse sistema ocorreu por meio de um terceiro componente especializado em executar a composição das skills providas pelas câmeras e processadores de imagens conectados à rede, cujo descobrimento acontece através de um servidor MES que dispõe de um serviço de descoberta local de servidores OPC UA baseada em multicast e DNS. A funcionalidade de P&P é demonstrada a medida que os dispositivos são descobertos pela rede OPC UA e podem ser instantaneamente utilizados, tal que o componente de composição de *skills* exerce a análise de capacidades do sistema e se reconfigura automaticamente.

Os resultados mostraram uma alta complexidade de implementação do projeto,

justificada por cerca de quatro mil linhas de código escritas pelo autor e 11 MB de dados objetos. Ainda, a análise de uso de recursos computacionais mostrou um pico de até 53 *threads* geradas para gerenciar a conexão dos servidores durante o processo de descobrimento e um uso de cerca de 8 MB de memória por servidores ociosos.

Quanto ao uso de rede, utilizando a ferramenta Wireshark, foi possível observar os processos de comunicação referentes ao descobrimento de servidores. Entretanto, os experimentos tanto em rede local quanto em rede distribuída, conectada por VPN, mostraram falhas de conectividade.

A abordagem utilizada pela biblioteca open62541, de utilizar mensagens multicast e resolução de endereços URI por DNS, desenvolvida por Profanter et al. (2017), não se mostrou capaz de atravessar sub-redes distribuídas, um dos requisitos de interoperabilidade de I4.0, utilizando máquinas virtuais em nuvem. Já as falhas de conexão em uma rede local podem estar atribuídas ao complexo gerenciamento de clientes para um servidor OPC UA se comunicando com outro, que corresponde às interfaces de código da própria biblioteca open62541 e àquelas implementadas por Profanter et al. (2021).

As análises e estudos realizados são de importância para a continuidade dos projetos de modelagem e referência da Indústria 4.0, bem como para a comunidade acadêmica propor novas soluções para o cenário das fábricas adaptáveis, sendo o modelo proposto certamente extensível para diferentes casos de uso de sistemas de produção autoadaptativos. Logo, para trabalhos futuros, propõe-se o uso de outras tecnologias de descobrimento de serviços que sejam aplicáveis em sistemas distribuídos, alinhados com outros conceitos relevantes, como computação em nuvem. Recomenda-se, por exemplo, a avaliação de roteadores utilizados em multicast, ou, o uso de arquitetura de redes Software Defined Network (SDN) para auxiliar o processo de descobrimento.

A modelagem proposta para a solução da detecção automatizada de marcadores fiduciais utilizando câmeras pode ser revisada, identificando melhorias no modelo de informação e maior reaproveitamento de recursos existentes, conforme as especificações do padrão OPC UA. Se faz necessário, também, a revisão dos códigos utilizados no projeto, buscando solucionar os problemas de conectividade observados.

Por fim, a aprendizagem trazida pelas referências utilizadas neste trabalho são de extrema relevância para a carreira de um engenheiro mecatrônico prestes a ingressar no mundo da indústria, em plena revolução tecnológica causada pelos avanços da eletrônica e da informática. Os modelos revisados no trabalho caracterizam o cenário futuro das tecnologias de produção, cabendo ao interesse dos setores industriais e de serviços de todo o mundo.

REFERÊNCIAS

ALEMANHA. Bundesministerium für Wirtschaft und Klimaschutz. Plattform Industrie 4.0. **Aspects of the research roadmap in application scenarios**. Berlim, abr. 2016. Disponível em: https://www.plattform-i40.de/IP/Redaktion/EN/Downloads/Publikation/aspects-of-the-research-roadmap.html. Acesso em: 13 jul. 2022.

ALEMANHA. Bundesministerium für Wirtschaft und Klimaschutz. Plattform Industrie 4.0. **Industrie 4.0 plug-and-produce for adaptable factories**: Example use case definition, models, and implementation. Berlim, jun. 2017. Disponível em: https://www.plattform-i40.de/IP/Redaktion/DE/Downloads/Publikation/Industrie-40-20Plug-and-Produce.html. Acesso em: 13 jul. 2022.

ALEMANHA. Bundesministerium für Wirtschaft und Klimaschutz. Plattform Industrie 4.0. **Leitbild für Industrie 4.0**. Berlim, maio 2019. Disponível em: https://www.plattform-i40.de/IP/Redaktion/DE/Downloads/Publikation/Leitbild-2030-f\%C3\%BCr-Industrie-4.0.html. Acesso em: 13 jul. 2022.

ALEMANHA. Bundesministerium für Wirtschaft und Klimaschutz. Plattform Industrie 4.0. **Details of the Asset Administration Shell - Part 1**: The exchange of information between partners in the value chain of Industrie 4.0. Berlim, nov. 2020. Disponível em: https://www.plattform-i40.de/IP/Redaktion/DE/Downloads/Publikation/Details_of_the_Asset_Administration_Shell_Part1_V3.html. Acesso em: 13 jul. 2022.

AMSTERS, R. et al. Evaluation of low-cost/high-accuracy indoor positioning systems. **International conference on advances in sensors, actuators, metering and sensing**, IARIA XPS, fev. 2019.

ANGELSMARK, O. et al. Knowledge representation for reconfigurable automation systems. **International conference on robotics and automation**. Roma, abr. 2007.

BACKHAUS, J.; ULRICH, M.; REINHART, G. Classification, modelling and mapping of skills in automated production systems. *In*: ZAEH, M. F. (ed.). **Enabling manufacturing competitiveness and economic sustainability**. Springer International Publishing, Cham, p. 85–89, out. 2014.

BAHETI, R.; GILL, H. Cyber-physical systems. *In*: SAMAD, T.; ANNASWAMY, A. (ed.). **The impact of control technology**. IEEE Control Systems Society, p. 161–166, 2011. Disponível em: www.ieeecss.org. Acesso em: 13 jul. 2022.

BJÖRKELUND, A. et al. Knowledge for intelligent industrial robots. *In*: Konidaris, g. (ed.). **AAAI spring symposium: Designing intelligent robots**. SS-12-02, mar. 2012.

CELOZZI, C. et al. A 6-DOF ARTag-based tracking system. **IEEE transactions on consumer electronics**, v. 56, n. 1, p. 203–210, 2010.

CHESHIRE, S.; KROCHMAL, M. **RFC6762**. Internet Engineering Task Force (IETF), 2013. Disponível em: https://www.rfc-editor.org/rfc/rfc6762.html. Acesso em: 13 jul. 2022.

CUTTING-DECELLE, A. et al. ISO 15531 MANDATE: A product-process-resource based approach for managing modularity in production management. **Concurrent engineering**, SAGE, v. 15, n. 2, p. 217–235, 2007. Disponível em: https://doi.org/10.1177/1063293X07079329.

DE SÁ, A. et al. Sistemas de controle via rede. *In*: MACÊDO, R. J. A.; FARINES, J.M. A. (ed.). **Projeto de sistemas distribuídos e de tempo real para automação**. EDUFBA, Salvador, p. 9–42, 2018. Disponível em: http://repositorio.ufba.br/ri/handle/ri/26164. Acesso em: 13 jul. 2022.

DEUTSCHES INSTITUT FÜR NORMUNG. **DIN SPEC 91345:2016-04**. Referenzarchitekturmodell Industrie 4.0 (RAMI 4.0). Berlim, abr. 2016. Disponível em: https://www.beuth.de/de/technische-regel/din-spec-91345/250940128. Acesso em: 13 jul. 2022.

DOROFEEV, K.; ZOITL, A. Skill-based engineering approach using OPC UA programs. **International conference on andustrial informatics**, IEEE, Porto, n. 16, p. 1098–1103, jul. 2018.

FERREIRA, P.; LOHSE, N. Configuration model for evolvable assembly systems. **Conference on assembly technologies and systems**. CIRP, p. 75–79, maio 2012.

GARRIDO-JURADO, S. et al. Generation of fiducial marker dictionaries using mixed integer linear programming. **Pattern recognition**, v. 51, out. 2015.

INTERNATIONAL ELECTROTECHNICAL COMMISSION. **IEC 62832-1:2020**. Industrial-process measurement, control and automation. Part 1: General principles. Genebra, out. 2020. Disponível em: https://webstore.iec.ch/publication/65858. Acesso em: 13 jul. 2022.

JAMMES, F.; SMIT, H. Service-oriented paradigms in industrial automation. **IEEE Transactions on Industrial Informatics**, v. 1, n. 1, p. 62–70, 2005.

LEITNER, S. H.; MAHNKE, W. OPC UA service-oriented architecture for industrial applications. **Softwaretechnik-Trends**, v. 26, n. 4, p. 61–66, nov. 2006.

LUHMANN, T. et al. **Close-range photogrammetry and 3D imaging**. Berlim, Boston: De Gruyter, 2019.

MAHNKE, W.; LEITNER, S.-H. **OPC unified architecture**. Springer-Verlag Berlin Heidelberg, 2009. Disponível em: https://doi.org/10.1007/978-3-540-68899-0. Acesso em: 13 jul. 2022.

MALEC, J. et al. Knowledge-based reconfiguration of automation systems. **IEEE international conference on automation science and engineering**. Scottsdale, AZ, p. 170–175, set. 2007.

MOURTZIS, D. et al. Cloud-based integrated shop-floor planning and control of manufacturing operations for mass customisation. **Procedia CIRP**, v. 33, p. 9–16, dez. 2015.

ONORI, M.; BARATA, J.; FREI, R. Evolvable assembly systems basic principles. *In*: SHEN, W. (ed.). **Information technology for balanced manufacturing systems**. Springer US, Boston, MA, p. 317–328, set. 2006.

OPC FOUNDATION. **OPC unified architecture part 1**: Overview and concepts. Scottsdale, 2017. OPC 10000-1, v. 1.04. Disponível em: https://opcfoundation.org/developer-tools/specifications-unified-architecture/part-1-overview-and-concepts. Acesso em: 13 jul. 2022.

OPC FOUNDATION. **OPC unified architecture part 7**: Profiles. Scottsdale, 2017. OPC 10000-7, v. 1.04. Disponível em: https://opcfoundation.org/developer-tools/specifications-unified-architecture/part-7-profiles/. Acesso em: 13 jul. 2022.

OPC FOUNDATION. **OPC unified architecture part 12**: Discovery and global services. Scottsdale, 2018. OPC 10000-12, v. 1.04. Disponível em: https://opcfoundation.org/developer-tools/specifications-unified-architecture/part-12-discovery-and-global-services/. Acesso em: 13 jul. 2022.

OPC FOUNDATION. **OPC unified architecture part 2**: Security model. Scottsdale, 2018. OPC 10000-2, v. 1.04. Disponível em: https://opcfoundation.org/developer-tools/specifications-unified-architecture/part-2-security-model/. Acesso em: 13 jul. 2022.

OPC FOUNDATION. **OPC unified architecture part 12**: Programs. Scottsdale, 2021. OPC 10000-10, v. 1.05.00. Disponível em: https://opcfoundation.org/developer-tools/specifications-unified-architecture/part-10-programs/. Acesso em: 13 jul. 2022.

OPC FOUNDATION. **OPC unified architecture part 4**: Services. Scottsdale, 2021. OPC 10000-4, v. 1.05.00. Disponível em: https://opcfoundation.org/developer-tools/specifications-unified-architecture/part-4-services/. Acesso em: 13 jul. 2022.

OPC FOUNDATION. **OPC unified architecture part 6**: Mappings. Scottsdale, 2021. OPC 10000-6, v. 1.05.00. Disponível em: https://opcfoundation.org/developer-tools/specifications-unified-architecture/part-6-mappings. Acesso em: 13 jul. 2022.

OPC FOUNDATION. **OPC unified architecture part 100**: Device information model. Scottsdale, 2022. OPC 10000-100, v. 1.03.1. Disponível em: https://opcfoundation.org/developer-tools/specifications-unified-architecture/part-100-device-information-model/. Acesso em: 13 jul. 2022.

OPC FOUNDATION. **OPC unified architecture part 3**: Address space model. Scottsdale, 2022. OPC 10000-3, v. 1.05.01. Disponível em: https://opcfoundation.org/developer-tools/specifications-unified-architecture/part-3-address-space-model/. Acesso em: 13 jul. 2022.

OPC FOUNDATION. **OPC unified architecture part 5**: Information model. Scottsdale, 2022. OPC 10000-5, v. 1.05.01. Disponível em: https://opcfoundation.org/developer-tools/specifications-unified-architecture/part-5-information-model/. Acesso em: 13 jul. 2022.

OPEN62541. **Documentation**. 2021. V. 1.2 release. Disponível em: https://open62541.org/doc/1.2/. Acesso em: 13 jul. 2022.

OPENCV. **Camera calibration with OpenCV**. 2022. V. 4.6.0. Disponível em: https://docs.opencv.org/4.6.0/d4/d94/tutorial_camera_calibration.html. Acesso em: 13 jul. 2022.

PAUKER, F. et al. A systematic approach to OPC UA information model design. **Procedia CIRP**, v. 57, p. 321–326, 2016. ISSN 2212-8271. Disponível em: https://www.sciencedirect.com/science/article/pii/S2212827116312100. Acesso em: 13 jul. 2022.

PFROMMER, J.; SCHLEIPEN, M.; BEYERER, J. PPRS: Production skills and their relation to product, process, and resource. **International conference on emerging technologies & factory automation**, IEEE, n. 18, set. 2013.

PFROMMER, J. et al. Plug & produce by modelling skills and service-oriented orchestration of reconfigurable manufacturing systems. **Automatisierungstechnik**, v. 63, p. 790–800, 2015.

PRESSMAN, R. S. **Engenharia de software**: uma abordagem profissional. Porto Alegre: Bookman, McGraw-Hill, 2011. v. 7.

PROFANTER, S. **How to create custom OPC UA information models**. 2019. Disponível em: https://opcua.rocks/custom-information-models/. Acesso em: 13 jul. 2022.

PROFANTER, S. **From modelling to execution**: OPC UA information model tutorial. 2020. Disponível em: https://opcua.rocks/from-modelling-to-execution-opc-ua-information-model-tutorial/. Acesso em: 13 jul. 2022.

PROFANTER, S. **From unpacking to plug & produce**: Flexible components integration for robots in Industry 4.0. Tese (dissertação) — Technische Universität München, Munique, 2021. Disponível em: http://mediatum.ub.tum.de/?id=1552194. Acesso em: 13 jul. 2022.

PROFANTER, S. et al. A hardware-agnostic OPC UA skill model for robot manipulators and tools. **International conference on emerging technologies and factory automation**, IEEE, n. 24, p. 1061–1068, 2019.

PROFANTER, S. et al. OPC UA for plug & produce: Automatic device discovery using LDS-ME. **International conference on emerging technologies and factory automation**, IEEE, Limassol, set. 2017.

PROFANTER, S. et al. A generic plug & produce system composed of semantic OPC UA skills. **Open journal of the industrial electronics society**, IEEE, v. 2, p. 128–141, jan. 2021.

REINHART, G. et al. Automatic configuration (plug & produce) of industrial ethernet networks. **International conference on industry applications**, IEEE, São Paulo, n. 9, nov. 2010.

ROMERO-RAMIREZ, F.; MUÑOZ-SALINAS, R.; MEDINA-CARNICER, R. Speeded up detection of squared fiducial markers. **Image and vision computing**, v. 76, jun. 2018.

SACOMANO, J. B. et al. **Indústria 4.0**: conceitos e fundamentos. São Paulo: Blucher, 2018.

VAZQUES, C. E.; SIMÕES, G. S. **Engenharia de requisitos**: software orientado ao negócio. São Paulo: Brasport Livros e Multimídia Ltda., 2016.

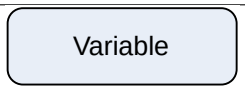
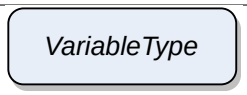
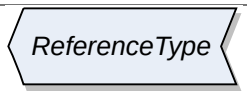
WAGNER, D.; SCHMALSTIEG, D. ARToolKitPlus for pose tracking on mobile devices. *In*: GRABNER, M.; GRABNER, H. (ed.). **Computer vision winter workshop**, Graz Technical University, jan. 2007.

APÊNDICE A – NOTAÇÃO GRÁFICA EM OPC UA

As notações gráficas definidas em OPC UA dão uma visão geral da estrutura do espaço de endereçamento e auxiliam na modelagem de gráfica informação. Cada classe de nó NodeClass possui sua própria representação gráfica, conforme mostrado no Quadro 3, em que todo tipo de classe é notado graficamente sombreado e, ainda, todo tipo abstrato é notado em itálico.

As referências entre nós são representadas por linhas, tal que a seta determina a direção da referência. Os tipos base de referências são mostrados no Quadro 4, qualquer outro tipo deve ser notado sobre a linha de uma referência simétrica, assimétrica ou hierárquica.

Quadro 3 – Notação de classes de nós

NodeClass	*Representação gráfica*	*Comentário*
Object		Pode conter a definição de tipo separada por ::, e.g., Variável1::Tipo1
ObjectType		Tipos abstratos são notados em itálico
Variable		Idem Object
VariableType		Idem ObjectType
DataType		Idem ObjectType
ReferenceType		Idem ObjectType
Method		—
View		—

Fonte: Mahnke e Leitner (2009, p. 328).

No diagrama de classes, todos ou alguns atributos de um nó podem ser especificados dentro do elemento gráfico, assim como as propriedades, que podem ser especificadas dentro ou fora, utilizando o elemento de referência HasProperty. A Figura 44 mostra um exemplo de modelagem de um dispositivo do tipo DeviceType de forma expandida, com blocos de tipo e variáveis referenciados por meio das setas, e de forma compacta, em que as propriedades são notadas dentro do bloco do objeto Device1.

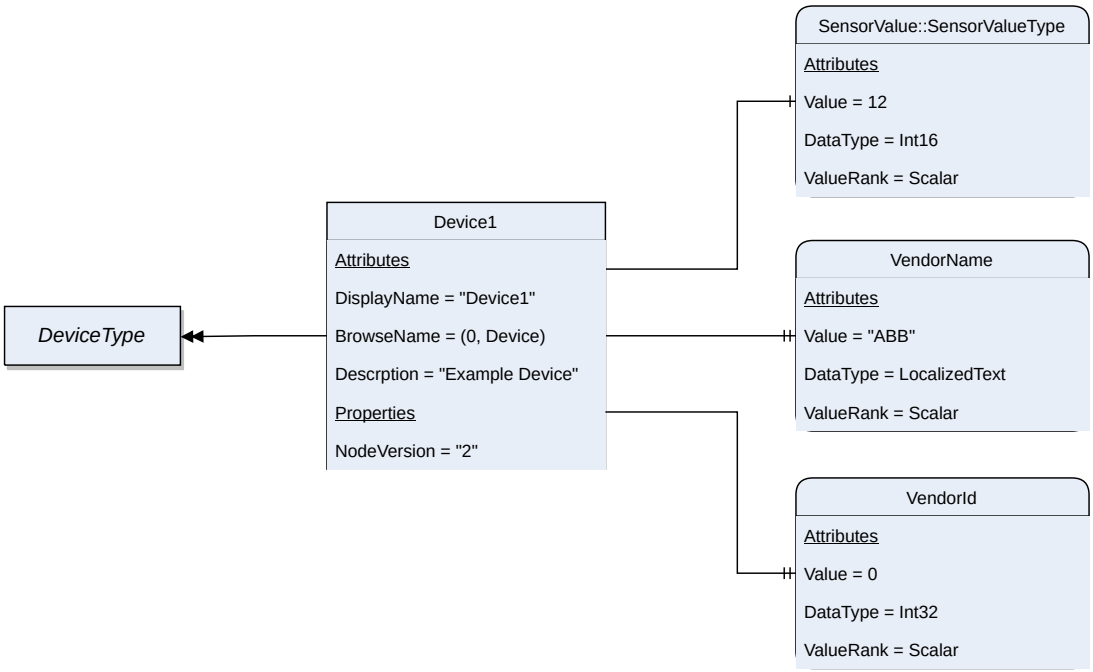
Quadro 4 – Notação de referências entre nós

ReferenceType	*Representação gráfica*
Qualquer tipo de referência simétrico	ReferenceType
Qualquer tipo de referência assimétrico	ReferenceType
Qualquer tipo de referência hierárquica	ReferenceType
HasComponent	
HasProperty	
HasTypeDefinition	
HasSubtype	
HasEventSource	

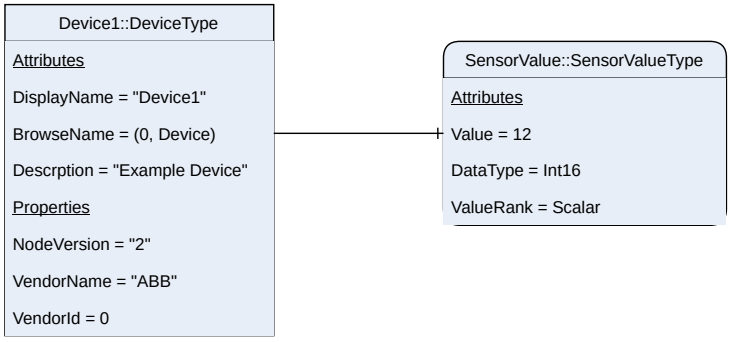
Fonte: Mahnke e Leitner (2009, p. 329).

Figura 44 – Exemplo de diagrama de classe

(a) Forma expandida.



(b) Forma compacta.



Fonte: Mahnke e Leitner (2009, p. 330 e 331).

APÊNDICE B – ATRIBUTOS DE NÓS EM OPC UA

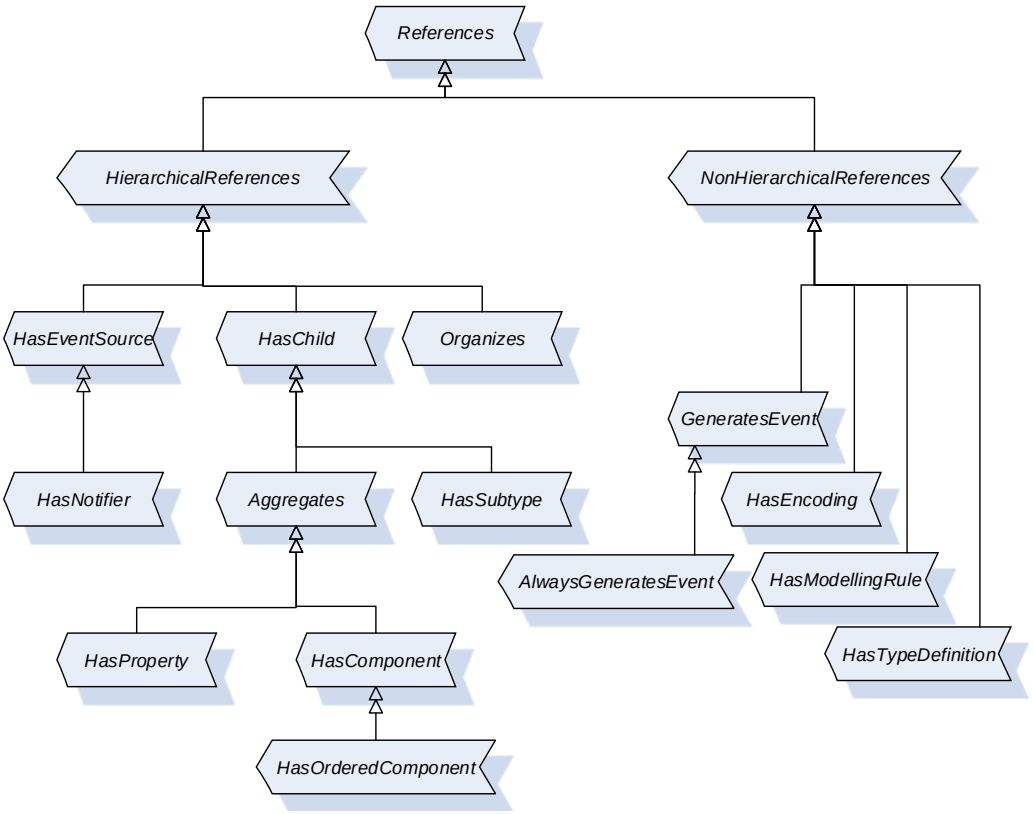
Quadro 5 – Lista de atributos de nós em OPC UA

Atributo	*ID*	*Descrição*
NodeId	1	Identificador único do nó no servidor.
NodeClass	2	O tipo base do nó.
BrowseName	3	Nome do nó em texto não localizado.
DisplayName	4	Nome do nó em texto localizado.
Description	5	Descrição do nó em texto localizado.
WriteMask	6	Indica quais atributos permitem escrita.
UserWriteMask	7	Indica quais atributos permitem escrita pelo usuário atual.
IsAbstract	8	Indica se o tipo de nó pode ou não ser instanciado.
Symmetric	9	Indica que as referências são iguais em abos os lados.
InverseName	10	O nome de busca (browse name) para uma referência inversa.
ContainsNoLoops	11	Indica que as a referência direta em um View não causa um loop.
EventNotifier	12	Indica que o nó pode ser usado para subscrição de eventos.
Value	13	Valor de uma variável.
DataType	14	NodeId do tipo de dado do valor da variável.
ValueRank	15	Número de dimensões de um valor.
ArrayDimensions	16	Tamanho de cada dimensão de um valor de um vetor.
AccessLevel	17	Descreve como o valor da variável pode ser acessado.
UserAccessLevel	18	Descreve como o valor da variável pode ser acessado considerando as permissões de acesso do usuário.
MinimumSamplingInterval	19	Especifica, em milisegundos, o período mínimo de amostragem de um valor pelo servidor.
Historizing	20	Especifica se o servidor está ativamente coletando dados históricos para a variável.
Executable	21	Define o método pode ser chamado.
UserExecutable	22	Define se o método pode ser chamado pelo usuário atual.

Fonte: Mahnke e Leitner (2009, p. 328).

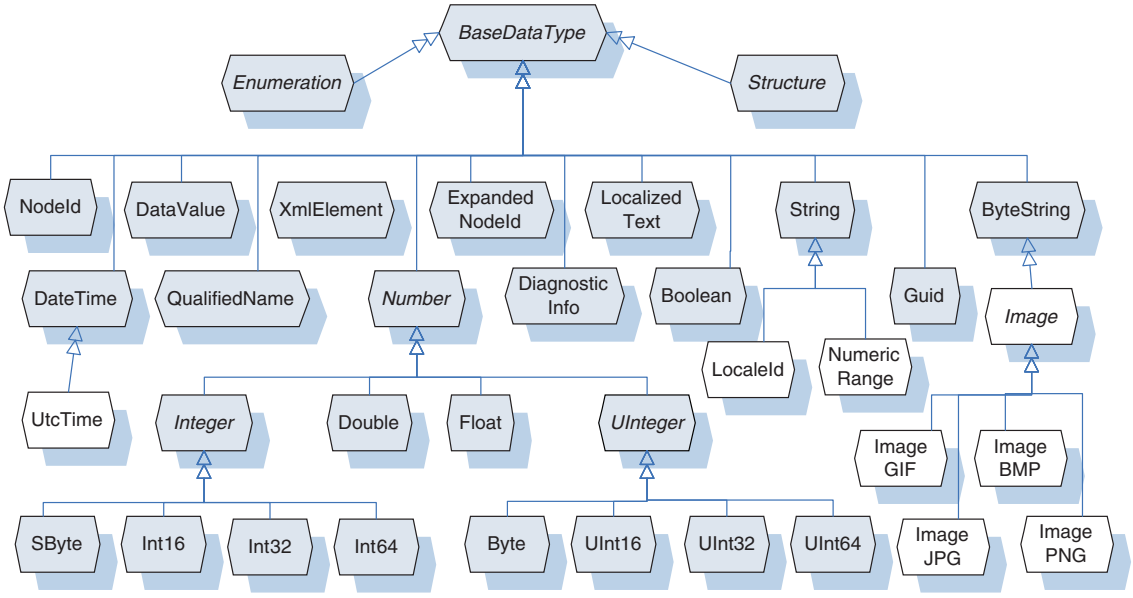
APÊNDICE C – HIERARQUIAS DE TIPOS EM OPC UA

Figura 45 – Hierarquia de tipos de referências padrões



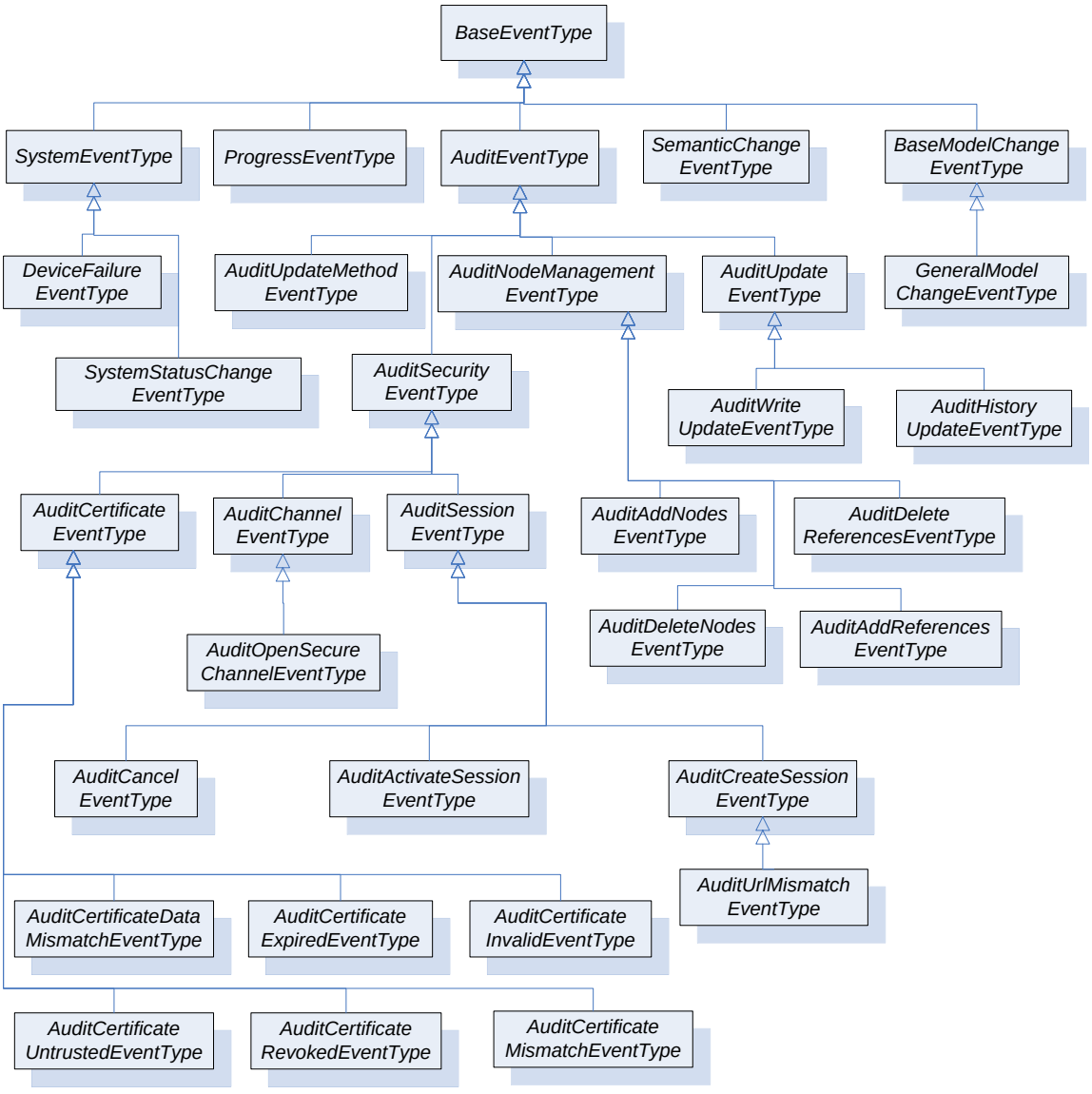
Fonte: OPC Foundation (2022b, p. 62).

Figura 46 – Hierarquia de tipos de dados padrões



Fonte: Mahnke e Leitner (2009, p. 335).

Figura 47 – Hierarquia de tipos de eventos padrões



Fonte: OPC Foundation (2022b, p. 85).