

**Matheus Ferreira da Silva**

**Desenvolvimento de uma Interface de  
Comunicação entre Servidores OPC DA e  
Clientes OPC UA**

Campina Grande, Brasil

Julho de 2019



**Matheus Ferreira da Silva**

**Desenvolvimento de uma Interface de Comunicação  
entre Servidores OPC DA e Clientes OPC UA**

Trabalho de Conclusão de Curso submetido à Coordenação de Graduação em Engenharia Elétrica da Universidade Federal de Campina Grande como parte dos requisitos necessários para a obtenção do grau de Engenheiro Eletricista.

Universidade Federal de Campina Grande - UFCG  
Centro de Engenharia Elétrica e Informática - CEEI  
Departamento de Engenharia Elétrica - DEE  
Coordenação de Graduação em Engenharia Elétrica - CGEE

Orientador: Professor Rafael Bezerra Correia Lima, D. Sc.

Campina Grande, Brasil

Julho de 2019



# Agradecimentos

Agradeço a Deus por tudo que me tem sido proporcionado durante essa jornada.

Agradeço a toda minha família, sem a qual jamais teria chegado onde cheguei. Em especial ao meu pai João Pedro e minha mãe Maria José pelo magnífico empenho que sempre tiveram para prover o necessário a todos os seus filhos. Às minhas irmãs Ana Maria e Ana Márcia, aos meus irmãos Márcio, Magno e Maxwelton, e à minha sobrinha Isabella por todos os bons momentos vividos e pelo apoio prestado.

Agradeço ao professor Rafael por toda a paciência e dedicação ao me orientar neste projeto. E a toda a equipe do Liec pela oportunidade de realizar este trabalho.

Agradeço a todos os bons amigos que fiz durante o curso, principalmente àqueles com quem compartilhei muitas tardes conversando e madrugadas estudando.

Por fim, agradeço a todos que de alguma forma me ajudaram a chegar até aqui.



# Resumo

O padrão OPC UA foi desenvolvido para substituir o OPC Clássico objetivando eliminar a dependência da plataforma *Windows* e permitir aos usuários lidar com dados e sistemas complexos. Entretanto, a implantação do padrão Clássico foi muito bem-sucedida, principalmente da especificação OPC DA. Em razão disso, o OPC UA foi projetado para permitir uma migração suave a partir do OPC Clássico. Uma das estratégias de migração consiste em usar aplicações que permitem a troca de informações entre os dois padrões. Neste trabalho foi desenvolvida uma aplicação do tipo *Wrapper* para permitir que um cliente OPC UA acesse dados fornecidos por um servidor OPC DA. O desenvolvimento foi realizado no IDE *Visual Studio* utilizando a plataforma .NET e SDKs específicos para OPC UA e OPC DA.

**Palavras-chave:** OPC DA, OPC UA, .NET, *Wrapper*.





# Abstract

The OPC UA standard was developed to replace Classic OPC to eliminate dependence on the Windows platform and allow users to handle complex data and systems. However, the implementation of the Classic standard was very successful, especially the OPC DA specification. Because of this, the OPC UA is designed to allow smooth migration from the Classic OPC. One of the migration strategies is to use applications that allow the exchange of information between the two standards. In this work, a Wrapper application was developed to allow an OPC UA client to access data provided by an OPC DA server. The development was performed in the Visual Studio IDE using the .NET platform and specific SDKs for OPC UA and OPC DA.

**Keywords:** OPC DA, OPC UA, .NET, *Wrapper*.



# Lista de ilustrações

Figura 1	– Representação dos objetos criados por um cliente OPC DA para acessar dados.	5
Figura 2	– Representação dos componentes estruturais fundamentais do OPC UA . . . .	7
Figura 3	– Representação do modelo de Nó do OPC UA. . . . .	8
Figura 4	– Modelo de Objeto OPC UA. . . . .	9
Figura 5	– Exemplo de um grupo de itens. . . . .	12
Figura 6	– Exemplo de características de um item. . . . .	12
Figura 7	– Trecho de código mostrando operação de leitura. . . . .	13
Figura 8	– Trecho de código mostrando operação de escrita. . . . .	13
Figura 9	– Trecho de código para a criação de <i>NodeIds</i> . . . . .	14
Figura 10	– Trecho de código mostrando a criação de uma pasta. . . . .	15
Figura 11	– Trecho de código para a criação de um objeto. . . . .	15
Figura 12	– Trecho de código para a criação de uma variável. . . . .	16
Figura 13	– Representação do espaço de endereçamento resultante. . . . .	16
Figura 14	– Características da variável criada. . . . .	16
Figura 15	– Tipo e valor da variável. . . . .	17
Figura 16	– Trecho de código que realiza a leitura de um valor. . . . .	17
Figura 17	– Trecho de código que realiza a escrita de um valor. . . . .	18
Figura 18	– Trecho de código que realiza a correspondência de tipos de DA para UA. . .	19
Figura 19	– Trecho de código que realiza a correspondência de tipos de UA para DA. . .	20
Figura 20	– Trecho de código que realiza o monitoramento dos Itens DA. . . . .	20
Figura 21	– Trecho de código que realiza o monitoramento das Variáveis UA. . . . .	21
Figura 22	– Função que atua caso haja mudança de valor em um Item DA. . . . .	22
Figura 23	– Função que atua caso haja mudança de valor em uma Variável UA. . . . .	23
Figura 24	– Tela inicial da aplicação. . . . .	26
Figura 25	– Tela de seleção de Itens. . . . .	26
Figura 26	– Tela de confirmação de seleção de um item. . . . .	27
Figura 27	– Conjunto de itens selecionados. . . . .	27
Figura 28	– Criação do servidor OPC UA. . . . .	28
Figura 29	– Conjunto de itens carregado a partir de uma arquivo XML. . . . .	28
Figura 30	– Representação do espaço de endereçamento resultante. . . . .	29
Figura 31	– Alteração no valor de um item do servidor DA. . . . .	30
Figura 32	– Leitura do valor da variável UA correspondente. . . . .	30
Figura 33	– Alteração no valor de uma variável servidor UA. . . . .	31

Figura 34 –Leitura do valor do item DA correspondente. . . . . 32

# Lista de abreviaturas e siglas

A & E	<i>Alarm &amp; Events</i>
API	<i>Application Program Interfaces</i>
CLP	Controlador Lógico Programável
COM	<i>Component Object Model</i>
DA	<i>Data Access</i>
DCOM	<i>Distributed COM</i>
DCS	<i>Distributed Control System</i>
ERP	<i>Enterprise Resource Planning</i>
HDA	<i>Historical Data Access</i>
HIM	<i>Human Machine Interface</i>
HTTP	<i>HyperText Transfer Protocol</i>
MES	<i>Manufacturing Execution Systems</i>
OLE	<i>Object Linking and Embedding</i>
OPC	<i>OLE for Process Control</i>
SCADA	<i>Supervisory Control and Data Acquisition</i>
SDK	<i>Software Development Kit</i>
UA	<i>Unified Architecture</i>
XML	<i>Extensible Markup Language</i>



# Sumário

<b>1</b>	<b>Introdução</b>	<b>1</b>
1.1	Objetivos	2
<b>2</b>	<b>Fundamentação Teórica</b>	<b>3</b>
2.1	O padrão OPC Clássico e a OPC <i>Foundation</i>	3
2.2	OPC <i>Data Access</i>	4
2.3	OPC <i>Unified Architecture</i>	5
2.3.1	Modelagem de informações: conceitos básicos	7
2.4	Migração do OPC Clássico para o OPC UA	9
<b>3</b>	<b>Materiais e Métodos</b>	<b>11</b>
3.1	OPC DA	12
3.1.1	Operações de leitura e escrita	13
3.2	OPC UA	14
3.2.1	Operações de leitura e escrita.	17
3.3	Integração entre OPC DA e OPC UA	17
3.3.1	Monitoramento dos valores dos itens e variáveis.	19
<b>4</b>	<b>Resultados</b>	<b>25</b>
4.1	Interface gráfica.	25
4.2	Testes de leitura e escrita.	29
<b>5</b>	<b>Conclusões</b>	<b>33</b>
	<b>Referências</b>	<b>35</b>





# 1 | Introdução

Nas últimas décadas as plantas industriais têm experimentado grandes avanços tecnológicos, sobretudo no que se refere aos sistemas de comunicação e automação industrial.

Grande parte dos avanços nos ambientes industriais desde a última década do século XX se deve à ampla difusão do uso de sistemas de automação baseados em software e, especialmente, ao emprego de PCs (*Personal Computers* – Computadores Pessoais) baseados no sistema operacional *Windows*. Entretanto, apesar desses fatores terem impulsionado fortemente o desenvolvimento tecnológico no ambiente industrial, a existência de um grande número de fabricantes de dispositivos voltados para aplicações de automação, cada qual com seus *drivers* específicos, tornou necessária a criação de um conjunto de especificações que permitissem o acesso padronizado aos dados de automação em sistemas baseados em *Windows*. Foi definido, então, o padrão OPC *Data Access* – OPC DA [1] .

Apesar da flexibilidade e das vantagens proporcionadas pelo OPC DA no que se refere à capacidade de trocar dados entre diferentes sistemas de automação industrial, o chamado OPC Clássico (composto pelo OPC DA e por outras variações do padrão OPC) tem algumas limitações importantes, principalmente no que diz respeito à dependência da plataforma *Windows* e à incapacidade de lidar com dados complexos. A fim de superar estas limitações, foi criado o OPC *Unified Architecture* – OPC UA [1].

O OPC UA pode ser visto como uma evolução do OPC Clássico, de modo que é possível a coexistência de ambos por meio de métodos que permitem a “tradução” entre os dois padrões, não sendo obrigatória a substituição da tecnologia clássica já implantada. Atualmente, milhares de produtos utilizam o OPC Clássico, e podem também utilizar as vantagens do OPC UA. A migração do padrão mais antigo para o mais moderno pode ser feita por meio do desenvolvimento de interfaces de comunicação entre eles [2].

## 1.1 Objetivos

Objetivo geral pretendido com a realização deste trabalho é implementar uma interface de comunicação entre servidores OPC DA e clientes OPC UA usando SDKs (*Software Development Kits*) e a linguagem de programação C#, visando proporcionar a coleta de dados de dispositivos compatíveis com OPC DA, bem como a disponibilização destes dados em uma interface OPC UA.

Para alcançar o objetivo proposto foram estabelecidos vários requisitos a serem atendidos, os quais são apresentados a seguir.

Requisitos estruturais:

- Implementar um cliente OPC DA capaz de se conectar com um servidor OPC DA;
- Criar um servidor OPC UA capaz de disponibilizar dados para Clientes OPC UA;
- Criar uma interface gráfica que permita ao usuário escolher um conjunto de itens de um servidor OPC DA para que estes sejam disponibilizados por meio de um servidor OPC UA;
- Criar um espaço de endereçamento no servidor OPC UA correspondente aos itens escolhidos pelo usuário para estabelecer a correspondência entre itens de um servidor OPC DA e variáveis de um servidor OPC UA;
- Permitir que o usuário salve um conjunto específico de itens em um arquivo XML para a configuração salva possa ser utilizada posteriormente.

Requisitos de intercâmbio de informações e sincronização de valores:

- Monitorar os valores de itens DA para detectar alterações;
- Ler o valor de um item após uma alteração;
- Realizar a adequação de informações do padrão OPC DA para o OPC UA;
- Escrever o valor adequado na variável correspondente;
- Monitorar os valores de variáveis UA para detectar alterações.
- Ler o valor de uma variável após uma mudança;
- Adequar o valor lido padrão OPC UA para OPC DA;
- Escrever o novo valor no respectivo item.

## 2 | Fundamentação Teórica

### 2.1 O padrão OPC Clássico e a OPC *Foundation*

Na década de 1990, a popularização do uso de computadores proporcionou grandes avanços no campo da automação industrial que, por sua vez, propiciaram o surgimento de uma grande variedade de dispositivos de automação fornecidos por diferentes fabricantes. Apesar da grande quantidade de opções a disposição dos usuários deste tipo de dispositivo, havia problemas relacionados à falta de padronização dos *drivers* dos dispositivos. Dessa forma, era necessário desenvolver *drivers* específicos que permitissem a comunicação entre diferentes componentes de um sistema. Além do mais, esse contexto dificultava bastante o uso de dispositivos de diferentes fabricantes em uma mesma aplicação [1].

Diante dessas circunstâncias, as companhias fornecedoras de aplicações de *software* voltadas para Interfaces Homem-Máquina (*Human Machine Interface – HMI*) e sistemas de Controle Supervisório e Aquisição de Dados (*Supervisory Control and Data Acquisition – SCADA*) enfrentavam, na época, dificuldades associadas à obtenção de dados de automação disponibilizados de variadas maneiras por dispositivos de diferentes fabricantes. Foi criada, então, em 1995, uma força-tarefa liderada por algumas companhias deste setor com o objetivo de definir um padrão para *drivers* de dispositivos a fim de proporcionar um acesso padronizado a dados em sistemas baseados em *Windows* [1].

O resultado obtido desse esforço conjunto foi a criação do OPC *Data Access – OPC DA* que foi a primeira especificação do padrão OPC (*OLE for Process Control*), que foi definido com base na tecnologia OLE (*Object Linking and Embedding*), da *Microsoft*.

Com o intuito de realizar a manutenção do padrão definido, foi instituída a organização sem fins lucrativos *OPC Foundation*, a qual foi capaz de definir e adotar práticas padrões relevantes muito mais rápido do que outras organizações. Uma das razões desse sucesso está no emprego das tecnologias COM (*Component Object Model*) e DCOM (*Distributed COM*) da *Microsoft*, as quais facilitaram a definição de APIs (*Application Program*

*Interfaces*). O foco em características importantes e o uso de tecnologias baseadas em *Windows* permitiram uma rápida adoção do padrão OPC para o caso de uso em questão [1].

Atualmente, sistemas SCADA, HMI, gerenciamento de processos, DCS (*Distributed Control System*), sistemas de controle baseados em PCs e MES (*Manufacturing Execution Systems* – Sistemas de Execução de Fabricação) devem suportar interfaces OPC. O padrão OPC é o único – universalmente aceito – que permite a troca de dados entre diferentes sistemas de automação industrial nas indústrias de fabricação e de processos [1].

Posteriormente à definição da especificação OPC DA, a qual trata da obtenção e disponibilização de dados em tempo real, surgiram outras especificações de acordo com os principais usos em aplicações de automação industrial. Foram criadas, então, outras duas importantes variações do padrão OPC, HDA (*Historical Data Access*) e A & E (*Alarm & Events*) para fornecer informações sobre dados históricos e sobre eventos, respectivamente. Este conjunto de especificações, bem como algumas outras de menor expressão, compõem o chamado OPC Clássico [1].

## 2.2 OPC Data Access

Embora o surgimento de outras variações tenha ajudado o padrão OPC a evoluir no sentido de permitir o fornecimento de informações mais complexas para determinadas aplicações, a especificação OPC DA é a principal dentre as que compõem o padrão clássico, sendo implementada em 99% dos produtos que usam a tecnologia OPC [1].

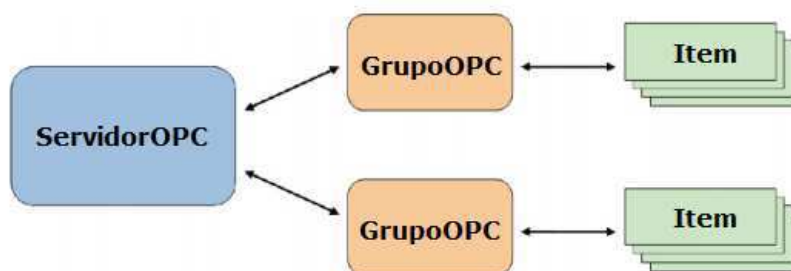
A interface OPC Data Access permite leitura, escrita e monitoramento de variáveis contendo dados de um processo corrente. O principal caso de uso é para transferir dados oriundos de CLPs, DCSs e outros dispositivos de controle para Interfaces Homem-Máquina e outros clientes capazes de exibir esses dados [1].

De modo geral, OPC é um padrão de comunicação que tem seu funcionamento baseado na interação entre servidores e clientes. Os servidores são responsáveis por disponibilizar dados e aguardar por requisições feitas por clientes. Uma vez que um servidor recebe uma solicitação, ele a responde e volta ao estado de espera. Em se tratando de OPC, é o cliente que decide quando e quais dados o servidor obterá dos sistemas subjacentes.

Clientes OPC DA selecionam explicitamente as variáveis (itens OPC) que querem ler, escrever ou monitorar no servidor. Os clientes OPC estabelecem uma conexão com o servidor por meio da criação de um objeto *ServidorOPC*. Este objeto oferece métodos que permitem navegar pelo espaço de endereçamento (coleção de informações que um

servidor torna visível para os seus clientes) hierarquicamente para encontrar itens e suas propriedades, tais como tipo do dado e direitos de acesso. Para acessar os dados, o cliente agrupa, em um objeto *GrupoOPC*, os itens com configurações idênticas como tempo de atualização (Figura 1) [1].

Figura 1 – Representação dos objetos criados por um cliente OPC DA para acessar dados.



Fonte: Adaptada de [1]

Em resumo, o objeto *ServidorOPC* mantém informações sobre o servidor e funciona como um contêiner para os objetos *GrupoOPC*. Estes, por sua vez, mantêm informações sobre si mesmos e fornecem mecanismos para conter e organizar logicamente os itens. Já os itens OPC representam conexões com fontes de dados no servidor. [1]

Por meio da especificação OPC DA são fornecidos dados de tempo real que podem não ser permanentemente acessíveis, por exemplo, quando a comunicação com um dispositivo é temporariamente interrompida. Para lidar com esta questão são fornecidos, além do valor do dado, dois outros parâmetros, o *Timestamp* (tempo exato em que o valor foi lido) e *Quality* (qualidade do valor fornecido). Este último especifica se o dado é preciso (*GOOD*), não disponível (*BAD*) ou desconhecido (*UNCERTAIN*) [1].

## 2.3 OPC Unified Architecture

A primeira e mais bem-sucedida especificação do OPC Clássico – OPC *Data Access* – foi projetada como uma interface para a comunicação de *drivers*, permitindo um acesso padronizado de leitura e escrita aos dados de tempo real vindos de diferentes dispositivos. O principal caso de uso foi o acesso a dados de automação, por parte de sistemas de Interface Homem-Máquina e SCADA, a partir de diferentes tipos de hardware e dispositivos de diferentes fabricantes [1].

Com o sucesso da adoção do OPC em milhares de produtos, ele passou a ser usado como uma interface padronizada entre diferentes sistemas de automação. Foi aplicado até mesmo em áreas para as quais ele não foi projetado. Além disso, havia ainda muitos

segmentos onde se desejava utilizar um padrão como o OPC, entretanto não era possível devido à sua dependência das tecnologias COM/DCOM, nativas do sistema operacional *Windows*. Somando-se à dependência da plataforma da *Microsoft*, houve, por parte das companhias associadas à *OPC Foundation*, solicitações para que fosse possível lidar com dados e sistemas complexos, a fim de superar as principais limitações do OPC Clássicos [1].

O padrão OPC *Unified Architecture* nasceu do desejo de criar um substituto efetivo para todas as especificações baseadas na tecnologia COM sem perder nenhum recurso e nem desempenho. Adicionalmente, esse substituto deveria cobrir todos os requisitos para interfaces de sistemas independentes de plataforma e com ricos e extensíveis recursos de modelagem, sendo capaz de descrever também sistemas complexos. Além disso, havia a necessidade de serem atendidos requisitos de escalabilidade, dada a ampla gama de aplicações do OPC, indo desde sistemas embarcados, passando por SCADA e DCS até MES e sistemas ERP (*Enterprise Resource Planning*) [1].

Para atender aos requisitos estabelecidos foi necessário garantir confiabilidade para a comunicação entre sistemas distribuídos, robustez, tolerância a erros, possibilidade de integrar interfaces OPC a sistemas executados em diferentes plataformas, alta performance em ambientes de *intranet*, possibilidade de comunicação via *internet* através de *firewalls*, segurança e controle de acesso, além de propiciar a interoperabilidade entre sistemas de diferentes fabricantes [1].

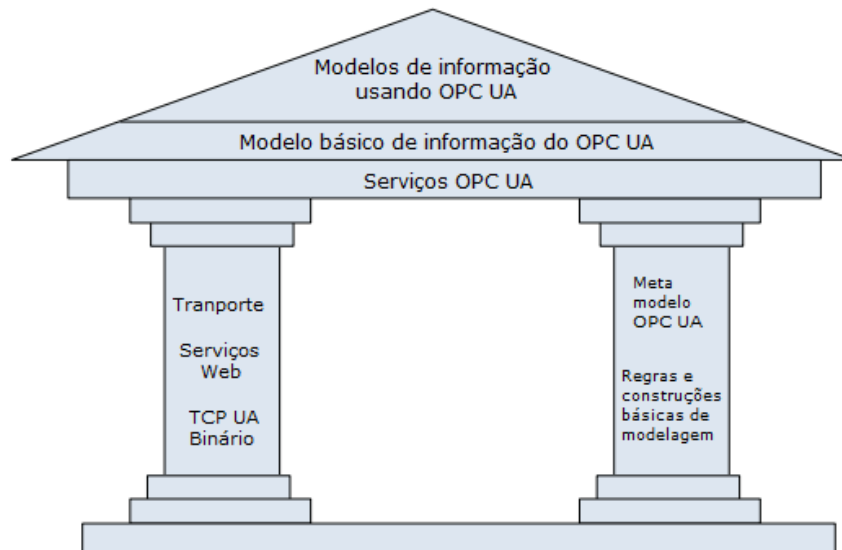
No que se refere aos requisitos para lidar com dados e sistemas complexos, foram adotadas medidas para tornar o OPC mais flexível e extensível, a exemplo do aprimoramento da modelagem de dados, disponibilização de métodos descritos por servidores e invocáveis por clientes, além do fornecimento de um modelo-base simples e abstrato a fim de permitir escalar de modelagens simples até modelos complexos [1].

Em síntese, a nova geração do OPC foi definida com base em requisitos e casos de uso especificados por organizações interessadas em utilizar o padrão OPC. Dessa forma, a *OPC Foundation* definiu como descrever e transportar dados e as organizações colaboradoras especificaram quais dados elas queriam que fossem descritos e transportados.

Os componentes fundamentais do OPC UA são os mecanismos de transporte e de modelagem de dados (Figura 2). Na parte de transporte são definidos diferentes mecanismos otimizados para diferentes casos de uso. Desde comunicação de alto desempenho via *intranet* usando o protocolo TCP binário até comunicação via *internet* usando Serviços *Web*, XML ou HTTP, por exemplo [1].

A modelagem de dados define as regras e os blocos de construção básicos necessários

Figura 2 – Representação dos componentes estruturais fundamentais do OPC UA



Fonte: Adaptada de [1]

para expor um modelo de informação usando OPC UA. São definidos também os pontos de entrada no espaço de endereçamento e os tipos básicos para construir uma hierarquia de tipos [1].

Os serviços UA são a interface entre servidores como supridores de um modelo de informação e os clientes como consumidores desse modelo. Os serviços são definidos de maneira abstrata. Eles usam os mecanismos de transporte para proporcionar a troca de dados entre clientes e servidores [1].

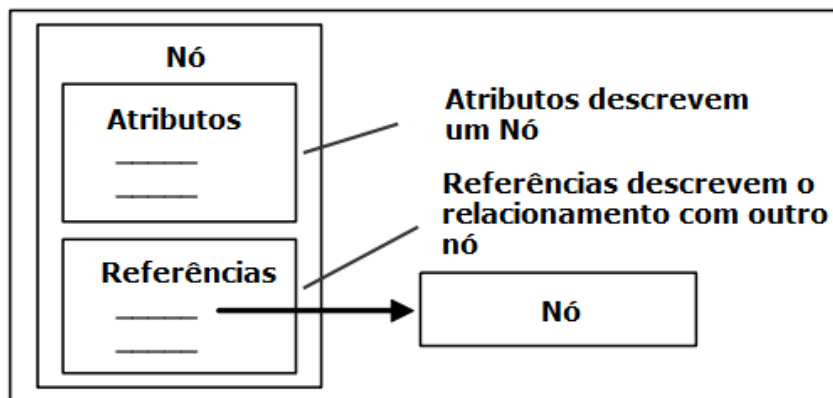
Para cobrir todos os recursos bem-sucedidos do OPC Clássico são definidos modelos de informação específicos pelo OPC UA sobre as especificações de base. Outras organizações também podem construir seus próprios modelos sobre a base UA ou a partir do modelo de informação OPC, expondo suas informações via OPC UA. Além disso, modelos de informação de fabricantes podem ser definidos usando diretamente a base UA, módulos OPC, ou outros modelos baseados em OPC UA [1].

### 2.3.1 Modelagem de informações: conceitos básicos

A modelagem de informações é empregada no OPC UA para construir um espaço de endereçamento (*AddressSpace*) que garante uma forma padrão de Servidores UA disponibilizarem informações para seus clientes. Um espaço de endereçamento é, portanto, o conjunto de informações disponibilizadas por um servidor que são visíveis para os seus clientes [3].

Os conceitos básicos de modelagem do OPC UA são Nós (*Nodes*) e Referências entre nós (*References*) (Figura 3). Os nós podem ser de diferentes classes (*NodeClasses*), dependendo do propósito a que se destinam. Existem nós que dizem respeito a instâncias e outros que representam tipos, por exemplo. A depender da classe de um nó ele pode ter um conjunto diferente de atributos (*Attributes*), que são os parâmetros usados para descrevê-lo. [4]. Por definição, uma referência descreve a relação entre exatamente dois nós. Portanto,

Figura 3 – Representação do modelo de Nó do OPC UA.



Fonte: Adaptada de [4]

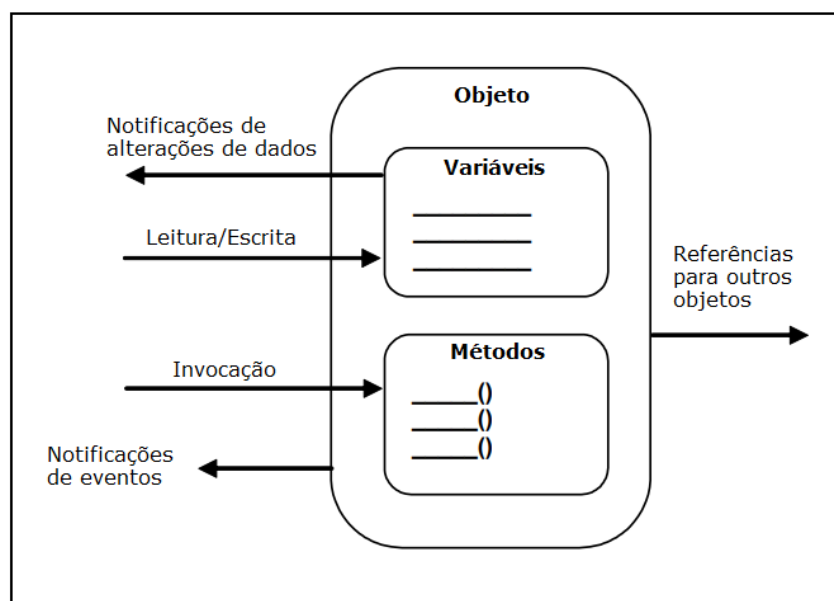
uma referência é unicamente identificada por um nó de origem, um nó de destino, pelo tipo da referência e por sua direção. Uma referência não pode ser acessada diretamente, apenas indiretamente, navegando a partir de um nó e, assim, seguindo as referências. Elas não são representadas como nós e não podem conter quaisquer atributos ou propriedades. No entanto, referências são usadas para expor diferentes semânticas sobre como os nós são conectados, podendo ser usadas, por exemplo, para explicitar uma hierarquia de tipos [1].

O principal objetivo de um Espaço de Endereçamento do OPC UA é fornecer uma forma padrão de servidores representarem Objetos (*Objects*) para clientes. Em virtude disso, o Modelo de Objeto (*Object Model*) do OPC UA foi projetado para atender a esse objetivo (Figura 4). Ele define Objetos em termos de Variáveis (*Variables*) e Métodos (*Methods*). Cada elemento diferente é representado por uma Classe de Nó (*NodeClass*) distinta. As Classes de Nós Objeto, Variável e Método são as mais importantes no OPC UA. De modo geral, objetos têm variáveis e métodos (conceitos conhecidos da programação orientada a objetos) e podem disparar eventos [4].

Os Nós da classe Objeto são usados para estruturar o espaço de endereçamento. Objetos não contêm outros dados além daqueles que descrevem o nó (atributos). Os valores dos dados referentes aos objetos são expostos usando Variáveis. Objetos podem ser utilizados para agrupar Variáveis, Métodos ou outros Objetos [1].



Figura 4 – Modelo de Objeto OPC UA.



Fonte: Adaptada de [4]

Um Nó da classe Variável representa um valor. O tipo de dado do valor depende da Variável. Clientes podem ler o valor, inscrever-se para receber alterações e modificar o valor em questão por meio de operações de escrita [1].

O Nós da classe Método representam um método, isto é, algo que é chamado por um cliente e retorna um resultado. Cada método especifica os argumentos de entrada que um cliente deve usar e o argumento de saída que se deve esperar como resultado [1].

## 2.4 Migração do OPC Clássico para o OPC UA

Um importante objetivo de projeto do OPC UA foi proporcionar uma fácil migração do OPC Clássico para a nova geração do OPC visando proteger o investimento no muito bem-sucedido padrão Clássico e aproveitar a grande base já instalada do OPC [1].

O OPC UA é bem mais flexível e tem muito mais recursos do que todas as especificações do OPC Clássico juntas. Além de ter incorporado todos os conceitos bem-sucedidos das especificações OPC existentes até então, o OPC UA corrigiu problemas conhecidos existentes no padrão Clássico e agregou padronização para muitos casos de uso adicionais [1].

O padrão UA permite um mapeamento simples e oferece estratégias de migração para integrar produtos OPC baseados nas especificações anteriores. Uma das estratégias

de migração não requer nenhuma mudança nos produtos existentes. Trata-se do uso de aplicações dos tipos *Wrapper* e *Proxy*, as quais funcionam como uma ponte entre o OPC Clássico e o OPC UA [5].

As soluções do tipo *Wrapper* permitem que clientes OPC UA acessem informações oriundas de um servidor OPC Clássico. Já os *Proxies*, permitem que clientes OPC DA, por exemplo, acessem informações disponibilizadas por servidores UA. Estas aplicações podem ser empregadas para tunelar o OPC Clássico através de *firewalls*, incluindo transmissão segura pela internet e acesso autenticado. Isto permite que usuários da tecnologia OPC Clássica possam desfrutar de recursos oferecidos pelo OPC UA e, assim, agrega valor às soluções já implantadas baseadas no padrão clássico [1].

## 3 | Materiais e Métodos

Com o objetivo de permitir que clientes OPC UA acessem dados disponibilizados por servidores OPC DA, foi criada uma aplicação que conta com recursos característicos de clientes OPC DA, bem como de servidores OPC UA. Os atributos de cliente DA são usados para coletar os dados oriundos de servidores DA e, de maneira análoga, os recursos próprios de servidores UA são empregados para disponibilizar adequadamente os dados a serem consumidos por clientes UA. Além dos componentes DA usados para adquirir os dados e da parte UA usada para disponibilizá-los, a aplicação desenvolvida conta também com uma parte destinada a realizar a “tradução” entre OPC DA e OPC UA, isto é, realizar a adequação dos dados ao padrão UA.

A construção da aplicação foi realizada no ambiente de desenvolvimento *Microsoft Visual Studio* utilizando-se a plataforma .NET (linguagem C) e *Kits* de Desenvolvimento de Software (*Software Development Kits* - SDKs). Neste ambiente foram integrados aspectos específicos de cada padrão OPC, além de ter sido construída uma interface gráfica e implementada a lógica de intercâmbio de informações entre o OPC DA e o OPC UA.

No que se refere ao conhecimento do assunto abordado e à familiarização com as ferramentas utilizadas, foi necessário realizar uma revisão bibliográfica acerca dos aspectos gerais e alguns aspectos específicos de ambos os padrões OPC, além do estudo dos materiais de referência dos SDKs e da plataforma de desenvolvimento utilizados. O uso de SDKs foi feito com o objetivo de facilitar e agilizar o desenvolvimento da aplicação por meio do conjunto de funções disponibilizadas, além dos materiais de referência e dos exemplos de código associados. Para criar o cliente OPC DA foi empregado o SDK “*OPC DA .NET Client Development Toolkit*” da companhia *Advosol* [6]. Já para a implementação do servidor OPC UA, foi utilizada a solução “*.NET Based OPC UA Client Server SDK*” oferecida pela *Unified Automation* [7]. Ambos os *kits* de desenvolvimento são específicos para a plataforma .NET.

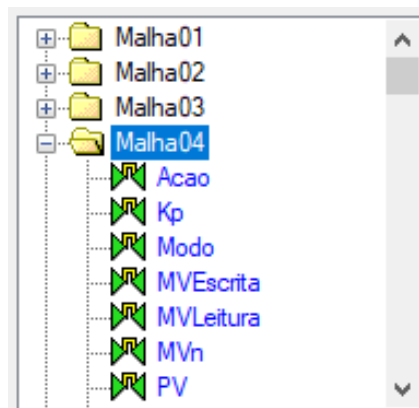
Para auxiliar no desenvolvimento e teste da aplicação foram empregadas as ferra-

mentas “*OPC DA V2 Test Client*” disponibilizada pela *Advosol* e “*UA Expert*” fornecida pela *Unified Automation*. Além disso, foi utilizado para fins de teste servidor OPC DA genérico “*LIEC.100Malhas.V1*” fornecido pelo LIEC (*Laboratório de Instrumentação Eletrônica e Controle*), onde foram realizadas as atividades deste projeto.

### 3.1 OPC DA

Tomando como base alguns exemplos do *Toolkit* da *Advosol*, foi implementado um cliente OPC DA capaz de obter dados disponibilizados por servidores DA. Essa etapa foi realizada com o objetivo de entender alguns aspectos importantes da interação entre clientes e servidores OPC DA, tais como as operações de escrita e leitura. Além disso, foi possível também compreender melhor como os dados são organizados com base nos conceitos de *Item* e *Grupo* de itens (Figura 5). Além de entender como é realizada a caracterização das informações, segundo a especificação OPC DA, por meio de variáveis que apresentam *Nome*, *Valor*, *Qualidade* do valor obtido e *Instante de tempo* de sua obtenção (Figura 6).

Figura 5 – Exemplo de um grupo de itens.



Fonte: Próprio Autor.

Figura 6 – Exemplo de características de um item.

Item	<input type="text" value="Malha04.Acao"/>	Value	<input type="text" value="0"/>
Access Rights	<input type="text" value="OPC_READWRITEAE"/>	Quality	<input type="text" value="GOOD"/>
		Timestamp	<input type="text" value="04/07/2019 03:57:35"/>

Fonte: Próprio Autor.

### 3.1.1 Operações de leitura e escrita

A função mais comum de um cliente OPC DA é ler o valor um item disponibilizado por servidor OPC DA por meio de uma operação de leitura. Entretanto, é possível também que um cliente estabeleça o valor de um determinado item mediante uma operação de escrita. Na Figura 7 é mostrado um exemplo de uma operação de leitura. Já na Figura 8 é mostra um exemplo de como é realizada a escrita de um valor.

Figura 7 – Trecho de código mostrando operação de leitura.

```
//-----//  
  
// Criação de uma estrutura para abrigar os Itens do servidor ao qual o cliente se conectará.  
OpcServer DAServer = new OpcServer();  
  
// Conexão com um servidor OPC DA.  
DAServer.Connect("Nome_do_Servidor");  
  
// Criação de grupo de leitura e escrita de valores.  
OPCDA.NET.SyncIOGroup ReadAndWriteGroup = new SyncIOGroup(DAServer);  
  
// Estrutura específica para receber o resultado da operação de leitura.  
OPCItemState Resultado;  
  
// Operação de Leitura.  
ReadAndWriteGroup.Read(OPCDATASOURCE.OPC_DS_DEVICE, "ItemId", out Resultado);  
// São especificados uma fonte de dados (DEVICE ou CACHE), o identificador de um Item e  
// um destino para o resultado da operação de leitura.  
  
// Valor lido de um Item.  
object Valor_do_Item = Resultado.DataValue;  
  
//-----//
```

Fonte: Próprio Autor.

Figura 8 – Trecho de código mostrando operação de escrita.

```
//-----//  
  
// Criação de uma estrutura para abrigar os Itens do servidor ao qual o cliente se conectará.  
OpcServer DAServer = new OpcServer();  
  
// Conexão com um servidor OPC DA.  
DAServer.Connect("Nome_do_Servidor");  
  
// Criação de grupo de leitura e escrita de valores.  
OPCDA.NET.SyncIOGroup ReadAndWriteGroup = new SyncIOGroup(DAServer);  
  
// Exemplo de valor a ser escrito.  
object valor = 0.0;  
  
// Operação de Leitura.  
ReadAndWriteGroup.Write("ItemID", valor);  
  
// São especificados o identificador de um Item e um valor a ser escrito.  
  
//-----//
```

Fonte: Próprio Autor.

## 3.2 OPC UA

O passo seguinte foi a análise e implementação de um servidor OPC UA simples tendo como base alguns dos exemplos disponibilizados junto ao *Toolkit* da *Unified Automation*. Nessa etapa buscou-se entender como se dá a estruturação de servidores OPC UA e aprender sobre elementos essenciais empregados na sua composição, tais como *Nós*, *Referências*, *Objetos* e *Variáveis*. Nessa parte do projeto foram utilizadas funções que permitem a criação e definição desses elementos básicos que compõem o padrão em questão.

Na Figura 9 são mostrados exemplos de como são criados os chamados “*NodeIds*”, os elementos básicos que são usados para identificar *nós* e, por isso, são essenciais para a definição e identificação de componentes mais complexos como *variáveis* e *objetos*.

Figura 9 – Trecho de código para a criação de *NodeIds*.

```
//-----//  
  
// Criação de um "NodeId" chamado "Exemplo1" usado para identificar um nó.  
// *A criação de um identificador para um nó subacente a criação do nó.  
    NodeId Exemplo1 = ParsedNodeId.Construct(0, "Exemplo1", DefaultNamespaceIndex);  
  
// Criação de um "NodeId" chamado "Exemplo2" usado para identificar um nó.  
    NodeId Exemplo2 = ParsedNodeId.Construct(0, "Exemplo2", DefaultNamespaceIndex);  
    class UnifiedAutomation.UaBase.NodeId  
// Definição: NodeId Construct(int baseType, string baseId, ushort namespaceIndex);  
// São especificados um tipo base, uma string de identificação e um namespace.  
  
//-----//
```

Fonte: Próprio Autor.

Na Figura 10 é mostrado o procedimento para a criação de um objeto capaz de agrupar outros objetos e que funciona, nesse caso, como uma pasta.

Na Figura 11 é mostrada a criação de um objeto destinado a agrupar variáveis.

Na Figura 12 é mostrada a criação de uma variável e como são definidos alguns de seus atributos.

Na Figura 13 é mostrado o espaço de endereçamento resultante da sequência de operações para a criação de objetos e variável. Essa representação foi obtida por meio do *UA Expert*.

Nas Figuras 14 e 15 são mostradas, respectivamente, características da variável criada e do seu valor.

Figura 10 – Trecho de código mostrando a criação de uma pasta.

```
//-----//
// Criação de um Objeto OPC UA.
// Este objeto funcionará como uma pasta capaz de "guardar" outros objetos.

// Criação de um nó necessário à definição do objeto.
NodeId FolderNode = ParsedNodeId.Construct(0, "Folder_Node", DefaultNamespaceIndex);

// Definição dos atributos do objeto a ser criado.
CreateObjectSettings Folder_settings = new CreateObjectSettings()
{
    //Todos os atributos são de tipos predefinidos adequados ao padrão OPC UA.
    ParentNodeId = UnifiedAutomation.UaBase.ObjectIds.ObjectsFolder, // Nó "pai".
    ReferenceTypeId = UnifiedAutomation.UaBase.ReferenceTypeIds.Organizes, // Tipo de referência.
    RequestedNodeId = FolderNode, // Nó do objeto.
    BrowseName = new QualifiedName("Pasta_Exemplo", DefaultNamespaceIndex), // Nome do objeto.
    TypeDefinitionId = UnifiedAutomation.UaBase.ObjectTypeIds.FolderType // Tipo do objeto.
};

// Criação do objeto propriamente dita.
CreateObject(Server.DefaultRequestContext, Folder_settings);
UnifiedAutomation.UaBase.ObjectNode BaseNodeManager.CreateObject(RequestContext context, CreateObjectSettings settings)
// Definição: ObjectNode CreateObject(RequestContext context, CreateObjectSettings settings);
// São especificados o contexto para o objeto e suas configurações.

//-----//
```

Fonte: Próprio Autor.

Figura 11 – Trecho de código para a criação de um objeto.

```
//-----//
// Criação de um Objeto OPC UA capaz de "guardar" variáveis.

// Criação de um nó necessário à definição do objeto.
NodeId ObjectNode = ParsedNodeId.Construct(0, "Objeto", DefaultNamespaceIndex);

// Definição dos atributos do objeto a ser criado.
CreateObjectSettings Object_settings = new CreateObjectSettings()
{
    //Todos os atributos são de tipos predefinidos adequados ao padrão OPC UA.
    ParentNodeId = FolderNode, // Nó "pai".
    ReferenceTypeId = UnifiedAutomation.UaBase.ReferenceTypeIds.Organizes, // Tipo de referência.
    RequestedNodeId = ObjectNode, // Nó do objeto.
    BrowseName = new QualifiedName("Objeto_Exemplo", DefaultNamespaceIndex), // Nome do objeto.
    TypeDefinitionId = UnifiedAutomation.UaBase.ObjectTypeIds.BaseObjectType // Tipo do objeto.
};

// Criação do objeto propriamente dita.
CreateObject(Server.DefaultRequestContext, Object_settings);
UnifiedAutomation.UaBase.ObjectNode BaseNodeManager.CreateObject(RequestContext context, CreateObjectSettings settings)
// São especificados o contexto para o objeto e suas configurações.

//-----//
```

Fonte: Próprio Autor.

Figura 12 – Trecho de código para a criação de uma variável.

```
//-----//
// Criação de uma variável.

// Criação de um nó necessário à definição da variável.
NodeId VariableNode = ParsedNodeId.Construct(0, "Variavel", DefaultNamespaceIndex);

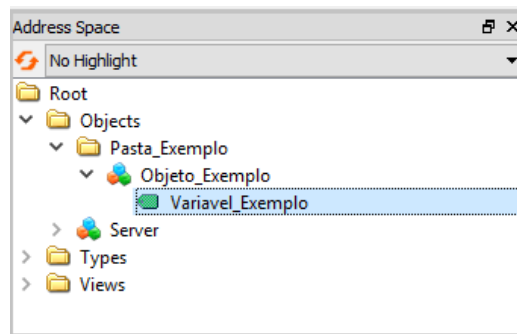
// Definição dos atributos da variável a ser criada.
CreateVariableSettings Variable_settings = new CreateVariableSettings()
{
    ParentNodeId = ObjectNode, // Nó "pai".
    ReferenceTypeId = UnifiedAutomation.UaBase.ReferenceTypeIds.HasComponent, // Tipo de referência.
    RequestedNodeId = VariableNode, // Nó do variável.
    BrowseName = new QualifiedName("Variavel_Exemplo", DefaultNamespaceIndex), // Nome da variável.
    TypeDefinitionId = UnifiedAutomation.UaBase.VariableTypeIds.DataItemType, // Tipo da variável.
    DataType = UnifiedAutomation.UaBase.DataTypeIds.Double, // Tipo do dado.
    AccessLevel = AccessLevels.CurrentReadOrWrite, // Níveis de acesso.
    Value = 0 // Valor.
};

CreateVariable(Server.DefaultRequestContext, Variable_settings);
UnifiedAutomation.UaBase.VariableNode BaseNodeManager.CreateVariable(RequestContext context, CreateVariableSettings settings)
// São especificados o contexto para a variável e suas configurações.

//-----//
```

Fonte: Próprio Autor.

Figura 13 – Representação do espaço de endereçamento resultante.



Fonte: Próprio Autor.

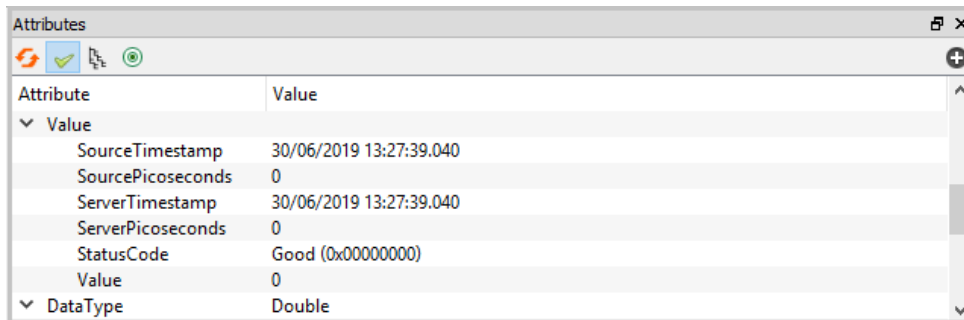
Figura 14 – Características da variável criada.

Attribute	Value
NodeId	ns=2;s=0:Variavel
NodeClass	Variable
BrowseName	2, "Variavel_Exemplo"
DisplayName	"" , "Variavel_Exemplo"
Description	"" , "A variable that contains live automation data."

Fonte: Próprio Autor.



Figura 15 – Tipo e valor da variável.



Attribute	Value
SourceTimestamp	30/06/2019 13:27:39.040
SourcePicoseconds	0
ServerTimestamp	30/06/2019 13:27:39.040
ServerPicoseconds	0
StatusCode	Good (0x00000000)
Value	0
DataType	Double

Fonte: Próprio Autor.

### 3.2.1 Operações de leitura e escrita.

Na Figura 16 são mostrados os procedimentos realizados para ler o valor de uma variável OPC UA. Já na Figura 17 é mostrado como se realiza a escrita de um valor em uma variável.

Figura 16 – Trecho de código que realiza a leitura de um valor.

```
//-----//
// Operação de leitura de um valor de uma variável.
// Criação de um identificador para indicar qual atributo será manipulado.
NodeAttributeHandle attributeHandle;

// Definição do identificador com base na especificação de uma variável e um de seus atributos.
GetNodeHandle(Server.DefaultRequestContext, VariableNodeID, Attributes.Value, out attributeHandle);

// Criação de uma estrutura para conter o valor lido.
DataValue valor_lido;
class UnifiedAutomation.UaBase.DataValue
valor_lido = ReadAttribute(Server.DefaultRequestContext, attributeHandle);

// São especificados um contexto e um identificador de atributo de uma variável.
//-----//
```

Fonte: Próprio Autor.

## 3.3 Integração entre OPC DA e OPC UA

Para construir um servidor OPC UA com uma estrutura análoga a de um determinado servidor OPC DA foi utilizada a estratégia de permitir que o usuário selecione um conjunto de itens de um servidor OPC DA qualquer, os quais ele deseja acessar por meio de uma interface UA. Este procedimento de seleção de itens é realizado por meio de uma

Figura 17 – Trecho de código que realiza a escrita de um valor.

```
//-----//  
// Operação de escrita de um valor em uma variável.  
// Criação de um identificador para indicar qual atributo será manipulado.  
NodeAttributeHandle attributeHandle;  
  
// Definição do identificador com base na especificação de uma variável e um de seus atributos.  
GetNodeHandle(Server.DefaultRequestContext, VariableNodeID, Attributes.Value, out attributeHandle);  
  
// Criação de uma estrutura para conter o valor a ser escrito.  
Variant valor = 0.0;  
  
struct UnifiedAutomation.UaBase.Variant  
// Operação de escrita.  
WriteAttribute(Server.DefaultRequestContext, attributeHandle, valor);  
  
// São especificados um contexto, um identificador de atributo e o valor a ser escrito.  
//-----//
```

Fonte: Próprio Autor.

interface que oferece algumas funcionalidades próprias de um cliente OPC DA. À medida que os itens são selecionados, o identificador (*ItemID*) de cada item é adicionado a uma lista e, uma vez terminada essa seleção, os identificadores são utilizados para construir um servidor OPC UA capaz de disponibilizar dados referentes aos itens previamente escolhidos. Tendo em vista que os identificadores dos itens são constituídos, de modo geral, pelo nome do conjunto (agrupamento de itens com características em comum) ao qual pertencem e pelo nome específico do item dentro deste conjunto. Como por exemplo, um item cujo identificador é “*Malha01.Mod0*” tem o nome “*Mod0*” dentro de um conjunto chamado “*Malha01*”. Foi adotado o procedimento de criar um objeto no ambiente UA em correspondência a cada conjunto de itens no ambiente DA. Além disso, para cada item de um determinado conjunto foi criada uma variável pertencente ao objeto correspondente a este conjunto. Dessa forma, considerando o exemplo citado, seria criado um objeto “*Malha01*” e uma variável “*Mod0*” associada a ele.

Uma vez solucionadas as questões de como obter os dados fornecidos por um servidor OPC DA e de como disponibilizá-los por meio de um servidor OPC UA, o passo seguinte foi voltado para a “tradução” das informações entre os ambientes DA e UA. Além da necessidade de criar uma estrutura de objetos e variáveis de acordo com as exigências de modelagem do padrão UA como foi exemplificado anteriormente, é preciso também definir tipos apropriados para os valores das variáveis. Além disso, é preciso realizar procedimentos de adequação dos valores obtidos de servidores DA para que eles possam ser atribuídos às variáveis.

Um ponto crucial no processo de criação das variáveis OPC UA corresponden-

tes aos respectivos Itens DA é a definição do tipo de dado que uma variável irá conter. Esse tipo é especificado pelo atributo “*DataType*” e precisa ser compatível com o padrão OPC UA. Para isso, ele é escolhido dentre os tipos pré-estabelecidos pertencentes ao conjunto “*UnifiedAutomation.UaBase.DataTypeIds*”. Dessa forma, objetivando a coerência entre os tipos dos dados DA e UA foi estabelecida uma regra de correspondência segundo a qual, se o valor de um Item DA for do tipo “*Double*”, por exemplo, o atributo “*DataType*” da Variável UA correspondente receberá o tipo “*UnifiedAutomation.UaBase.DataTypeIds.Double*”. Da mesma forma é realizada a correspondência entre “*String*” e “*UnifiedAutomation.UaBase.DataTypeIds.String*”. A mesma estratégia é utilizada para outros dos principais tipos de dados comumente usados como “*Int16*”, “*Int32*”, “*Int64*”, “*Boolean*”, entre outros.

No que se aos valores dos dados propriamente ditos, também são necessários alguns procedimentos para adequá-los quando são passados de um padrão para outro. Quando os dados vêm do padrão DA eles são do tipo “*object*”, mas para que eles sejam escritos como valor de uma variável UA é preciso que eles sejam do tipo “*UnifiedAutomation.UaBase.Variant*”. Para isso é feita a conversão mostrada na Figura 18. Quando os dados fazem o caminho inverso, eles são originalmente do tipo “*UnifiedAutomation.UaBase.DataValue*” e para serem escritos como valor de um item DA é preciso que eles sejam do tipo “*object*” e, para isso, é realizado o procedimento mostrado na Figura 19.

Figura 18 – Trecho de código que realiza a correspondência de tipos de DA para UA.

```
//-----//
// Adequação de dados do padrão OPC DA para UA
// Dado vindo de um servidor DA.
object valor_DA = "0.0"; // Pode ser um valor 'Double', 'String', entre outros.
// Procedimento de construção de Informação de Tipo a respeito do dado.
// Informação necessária para a criação de uma "Variant" a partir de um "object".
UnifiedAutomation.UaBase.TypeInfo typeInfo = UnifiedAutomation.UaBase.TypeInfo.Construct(valor_DA);
// Dado adequado para operação de escrita no ambiente UA.
Variant valor_UA = new Variant(valor_DA, typeInfo);
struct UnifiedAutomation.UaBase.Variant
//-----//
```

Fonte: Próprio Autor.

### 3.3.1 Monitoramento dos valores dos itens e variáveis.

A fim de garantir que o servidor UA implementado forneça cópias fiéis dos valores providos por um determinado servidor DA, uma parte da aplicação destina-se a monitorar

Figura 19 – Trecho de código que realiza a correspondência de tipos de UA para DA.

```
//-----//
// Adequação de dados do padrão OPC UA para DA
// Exemplo de dado do tipo " UnifiedAutomation.UaBase.DataValue".
DataValue dataValue_UA = new DataValue();           //Atribuição genérica (ilustrativa).
Variant valor_UA = dataValue_UA.WrappedValue;     //"Variant" que compõe um "DataValue".
// Dado adequado para operação de escrita no ambiente DA.
object valor_para_DA = valor_UA.Value;            //"object" que compõe uma "Variant".
//-----//
```

object Variant.Value { get; }

Fonte: Próprio Autor.

os valores tanto dos itens OPC DA quanto das variáveis OPC UA. Nos dois casos (ambientes DA e UA) foram utilizadas instâncias de classes destinadas a tratar eventos de mudanças ocorridas nos dados. Para isso foram utilizadas ferramentas “*DataChangeEventHandler*” mostradas nas Figuras 20 e 21.

Figura 20 – Trecho de código que realiza o monitoramento dos Itens DA.

```
//-----//
// Monitoramento de mudança de valores dos Itens OPC DA
OPCDA.NET.DataChangeEventHandler dataChangeHandler_DA;
//-----//
```

delegate void OPCDA.NET.DataChangeEventHandler(object sender, OPCDA.NET.DataChangeEventArgs e)

```
// Chamada da função "DataChangeHandler_DA" para atuar sobre a mudança dos valores.
dataChangeHandler_DA = new OPCDA.NET.DataChangeEventHandler(this.DataChangeHandler_DA);
// O uso dessa estratégia se deu com base em uma exemplo fornecido pelo Toolkit da ADVOSOL
//-----//
```

Fonte: Próprio Autor.

Dessa forma, quando o valor de um determinado item sofre uma alteração no ambiente DA, essa mudança é detectada e o novo valor é atribuído à variável correspondente no ambiente UA por meio de uma operação de escrita destinada a alterar o atributo “*Value*” (Valor) da variável em questão. De maneira análoga, ao ser detectada uma alteração no valor de uma variável no ambiente UA provocada, por exemplo, por uma ação de escrita realizada por um cliente OPC UA, o novo valor da variável em questão é atribuído também ao item correspondente do espaço de endereçamento DA por meio de uma operação de escrita.

No caso do OPC DA, sempre que são detectadas mudanças nos valores dos itens é

Figura 21 – Trecho de código que realiza o monitoramento das Variáveis UA.

```
//-----//  
  
// Monitoramento de mudança de valores dos Itens OPC UA  
UnifiedAutomation.UaServer.DataChangeEventHandler dataChangeEventHandler_UA;  
  
// Chamada da função "DataChangeHandler_UA" para atuar sobre a mudança dos valores.  
dataChangeEventHandler_UA = new UnifiedAutomation.UaServer.DataChangeEventHandler(this.DataChangeHandler_UA);  
  
// Nesse caso, para cada item, "dataChangeEventHandler_UA" é passado como argumento para a função -->  
// -->"StartDataMonitoring()", a qual exige que se especifique também a variável a ser monitorada, -->  
// --> o atributo a ser monitorado, o modo de monitoramento, entre outros. Ver documentação.  
  
//-----//
```

Fonte: Próprio Autor.

invocada uma função (“*DataChangeHandler\_DA*” - Figura 22) destinada a verificar qual item sofreu alteração, realizar a adequação do novo valor ao padrão UA e cuidar para que o valor adequado seja escrito na variável UA correspondente. Já no caso do OPC UA são criados itens monitorados (“*MonitoredItems*”) para que, sempre que haja alteração em uma variável, seja chamada uma função (“*DataChangeHandler\_UA*” - Figura 23) encarregada de tornar o novo valor compatível com o padrão DA e realizar a escrita deste valor no item DA correspondente.

Um ponto importante a ser comentado sobre esse intercâmbio de valores entre os servidores DA e UA é que deve-se atentar para o fato de que toda vez que é detectada uma mudança no valor de um item DA e este é escrito em sua variável UA correspondente ocorre, então, uma mudança no valor da variável, afinal, o seu valor passou por uma operação de escrita. Dessa forma, o mecanismo responsável pelo monitoramento de valores no ambiente UA detecta essa mudança e tenta escrever este valor de volta no servidor OPC DA. Uma situação análoga acontece quando é detectada inicialmente uma mudança numa variável UA. A fim de evitar que os mecanismos de monitoramento e escrita de valores atuem repetidamente diante da situação citada anteriormente, foi empregada a estratégia de utilizar *flags* para que o mecanismo de escrita da parte DA possa indicar que uma alteração foi realizada por ele e evitar que o mecanismo da parte UA tente escrever de volta o valor. A mesma lógica se aplica à situação inversa. A estratégia adotada é mostrada também nas Figuras 22 e 23.

Figura 22 – Função que atua caso haja mudança de valor em um Item DA.

```
public void DataChangeHandler_DA(object sender, DataEventArgs e)
{
    for (int i = 0; i < e.sts.Length; ++i)    // Para cada Item monitorado.
    {
        //Determinação do identificador e do valor de um item com base nos dados fornecidos pelo evento "e".
        OPCItemState ItemState = e.sts[i];
        int handle = ItemState.HandleClient;           //"handle" correspondente ao item.
        OPCDA.NET.ItemDef item = ReadWriteGroup.FindClientHandle(handle);
        string ItemId = item.OpcIDef.ItemID;         // Identificador.
        object Valor_DA = item.OpcIRslt.DataValue;   // Valor.

        //Para a correspondência entre "ItemID's" e "NodeID's" foi criado um Dicionário:
        //Dictionary<String, NodeId> name_VariNode = new Dictionary<string, NodeId>();
        NodeId varinode = name_VariNode [ItemId];

        //Para a correspondência entre flags e "NodeID's" foram criados os Dicionários:
        //Dictionary<NodeId, int> flag_change_DA = new Dictionary<NodeId, int>();
        //Dictionary<NodeId, int> flag_change_UA = new Dictionary<NodeId, int>();

        if (flag_change_UA[varinode] == 0) // Se não houve escrita anterior na parte UA.
        {
            UnifiedAutomation.UaBase.TypeInfo tipo = UnifiedAutomation.UaBase.TypeInfo.Construct(Valor_DA);
            Variant valor_UA = new Variant(Valor_DA, tipo);           // Valor compatível com UA.
            NodeAttributeHandle nodehand;
            GetNodeHandle(Server.DefaultRequestContext, varinode, Attributes.Value, out nodehand);

            flag_change_DA[varinode] = 1; // Informa que foi realizada uma escrita pela parte DA.
            WriteAttribute(Server.DefaultRequestContext, nodehand, valor_UA);
        }

        else if (flag_change_UA[varinode] == 1) // Se houve escrita anterior na parte UA.
        {
            flag_change_UA[varinode] = 0;
            // Tendo-se ciência da atuação da parte UA, zera-se a sua flag para esperar por novas mudanças.
        }
    }
}
```

Fonte: Próprio Autor.

Figura 23 – Função que atua caso haja mudança de valor em uma Variável UA.

```
//-----//
public void DataChangeHandler_UA(RequestContext context, MonitoredItemHandle itemHandle, DataValue dataValue,
                                bool doNotBlockThread)
{
    object Valor_DA = dataValue.WrappedValue.Value;    // "dataValue" = valor da variável alterada.
    NodeId node = itemHandle.NodeId;                 // NodeId da variável alterada.

    // Foram utilizados os Dicionários:
    // Dictionary<NodeId, String> name_NodeVari = new Dictionary<NodeId, string>();
    // Dictionary<NodeId, int> flag_change_DA = new Dictionary<NodeId, int>();
    // Dictionary<NodeId, int> flag_change_UA = new Dictionary<NodeId, int>();

    String Nome_Item = name_NodeVari[node];

    if (flag_change_DA[node] == 0)                   // Se não houve escrita anterior na parte DA.
    {
        ReadWriteGroup.Write(Nome_Item, Valor_DA);
        flag_change_UA[node] = 1;                    // Informa que foi realizada uma escrita pela parte UA.
    }
    else if (flag_change_DA[node] == 1)              // Se houve escrita anterior na parte DA.
    {
        flag_change_DA[node] = 0;
        // Tendo-se ciência da atuação da parte DA, zera-se a sua flag para esperar por novas mudanças.
    }
}
//-----//
```

Fonte: Próprio Autor.





## 4 | Resultados

Como resultado da metodologia aplicada foi criada uma aplicação que permite que dados disponibilizados por um servidor OPC DA sejam acessados por meio de uma interface UA. Em síntese, a solução implementada comporta-se como um servidor OPC UA simples que tem um servidor DA como fonte dos dados que disponibiliza. Dessa forma, uma vez estabelecida a conexão entre aplicação desenvolvida e um servidor OPC DA é fornecida uma interface UA que permite que um cliente OPC UA comunique-se, por meio da aplicação, com um servidor DA. Esta conexão, uma vez estabelecida, proporciona não só a obtenção de dados por parte do cliente, mas também permite que este realize alterações nos valores dos itens do servidor OPC DA.

### 4.1 Interface gráfica.

Na fase inicial de execução da aplicação é exibida uma interface gráfica, mostrada na Figura 24, que possibilita ao usuário escolher um servidor OPC DA dentre aqueles instalados no computador em uso e, posteriormente, escolher os itens de seu interesse.

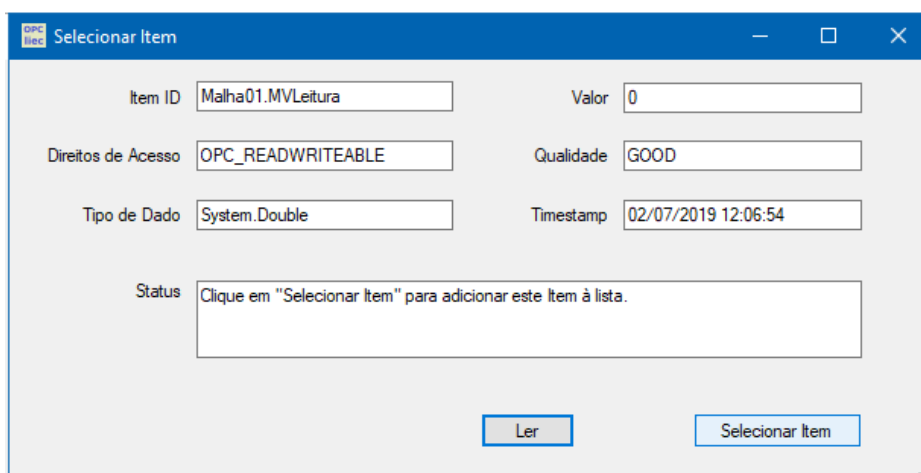
Na tela inicial da interface, clicando em “Selecionar”, o usuário é direcionado para uma outra tela que permite que ele escolha um servidor OPC DA e realize sua conexão por meio do botão “Conectar”. Após isso, são exibidos os itens do servidor organizados em pastas (Figura 25).

Para realizar a seleção de um item preciso clicar sobre ele. Ao fazer isso é aberta uma janela que possibilita a visualização de informações do item a partir da opção “Ler” ou a efetivação da escolha do item por meio do botão “Selecionar Item” (Figura 26).

Uma vez confirmada a seleção de um item, ele é adicionado a uma lista que pode ser visualizada na tela anterior (Figura 27). Os itens componentes desta lista são aqueles que poderão ser acessados por meio de uma interface UA. Uma vez selecionado um conjunto de itens de interesse, basta clicar em “Concluir”. O usuário pode também salvar esta

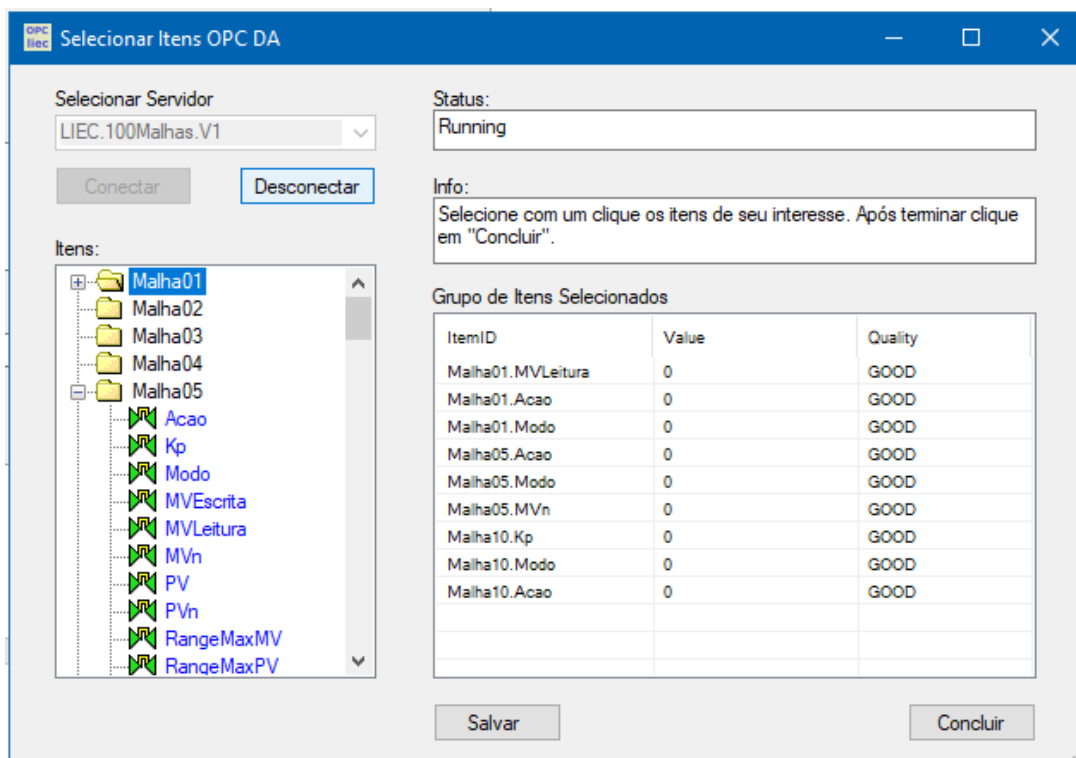


Figura 26 – Tela de confirmação de seleção de um item.



Fonte: Próprio Autor.

Figura 27 – Conjunto de itens selecionados.

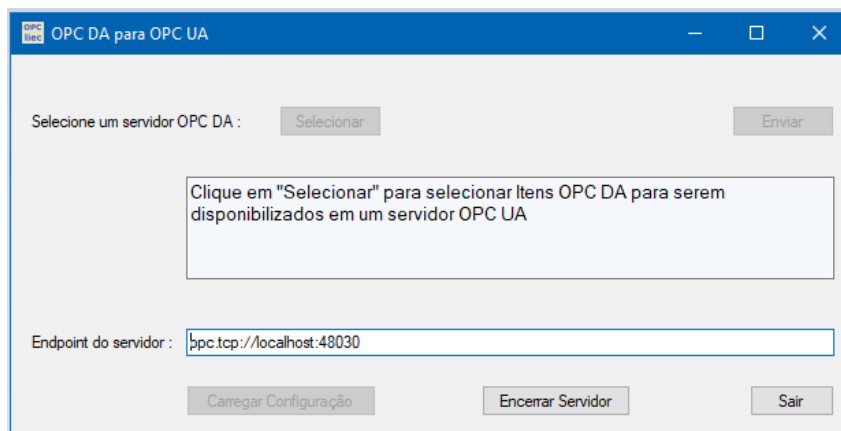


Fonte: Próprio Autor.

OPA UA. Ao ser criado o servidor UA, é mostrado seu *endpoint* na tela (Figura 28).

O usuário pode ainda carregar um conjunto de itens a partir de um arquivo XML clicando em “Carregar Configuração”. Na Figura 29 é mostrada uma configuração salva

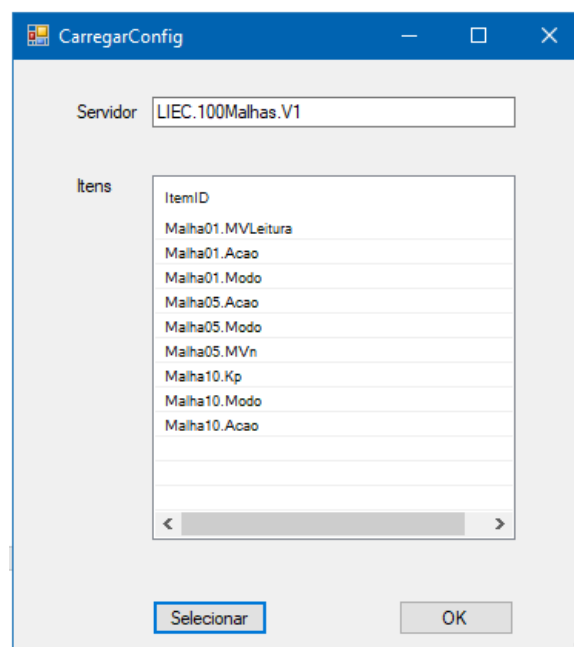
Figura 28 – Criação do servidor OPC UA.



Fonte: Próprio Autor.

em um arquivo XML. Esse recurso permite ao usuário salvar um conjunto de itens de seu interesse por meio da opção “Salvar” mostrada na Figura 27 e, posteriormente, reutilizar esta definição.

Figura 29 – Conjunto de itens carregado a partir de uma arquivo XML.



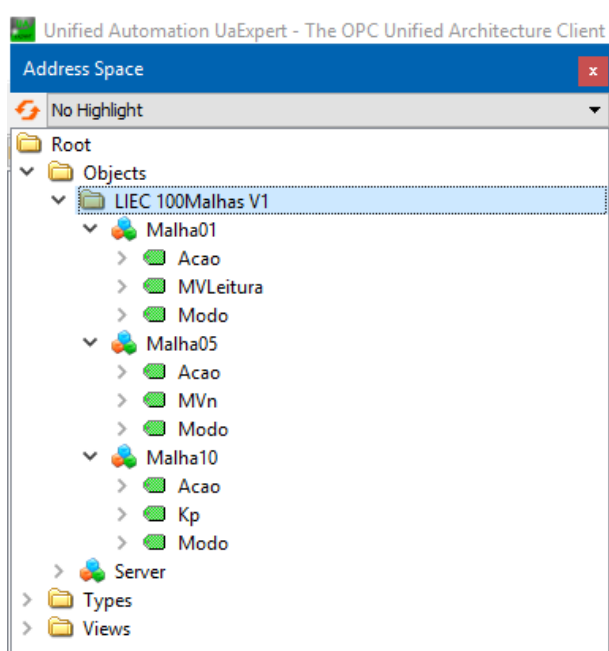
Fonte: Próprio Autor.

Realizados todos os procedimentos anteriormente descritos, a aplicação estará conectada ao servidor DA escolhido e pronta para ser conectada a um cliente OPC UA, ao qual proporcionará o acesso aos valores dos Itens DA selecionados, tanto para leitura

quanto para escrita. Neste estágio, o usuário pode escolher na tela inicial a opção “Encerrar Servidor”, cuja função é encerrar o servidor atual e reiniciar a aplicação, permitindo a criação de um novo servidor.

Na Figura 30 é mostrada uma captura de tela em que é possível visualizar, por meio do *software* UA *EXpert* da *Unified Automation*, o espaço de endereçamento resultante das operações realizadas anteriormente para criar um servidor OPC UA.

Figura 30 – Representação do espaço de endereçamento resultante.



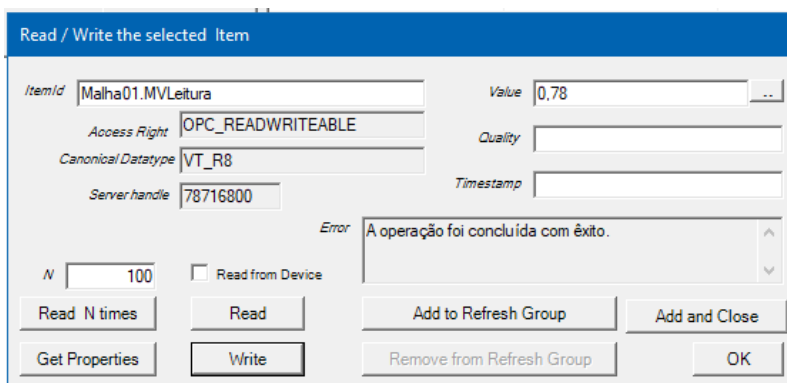
Fonte: Próprio Autor.

## 4.2 Testes de leitura e escrita.

A fim de testar a capacidade de sincronização de valores entre os itens DA e as suas variáveis UA correspondentes foram utilizadas a ferramenta de teste “*OPC DA V2 Test Client*” da *Advosol* e a ferramenta de teste e análise “*UA Expert*” fornecida pela *Unified Automation*. A primeira funciona como um cliente DA e foi conectada ao servidor DA. Já a segunda funciona como um cliente UA e foi conectada ao servidor UA.

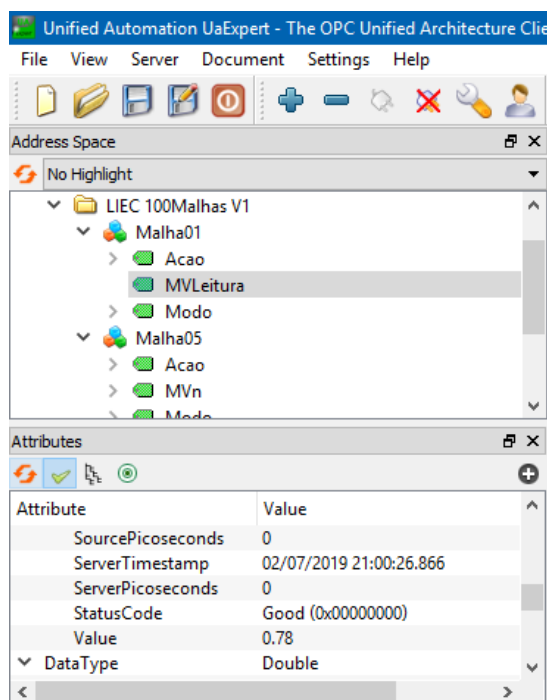
Inicialmente foram realizadas alterações no valor de um item DA por meio de uma operação de escrita realizada pelo cliente DA de teste (Figura 31). Em seguida foi lido o valor da variável correspondente no ambiente UA e o resultado pode ser visto na Figura 32.

Figura 31 – Alteração no valor de um item do servidor DA.



Fonte: Próprio Autor.

Figura 32 – Leitura do valor da variável UA correspondente.

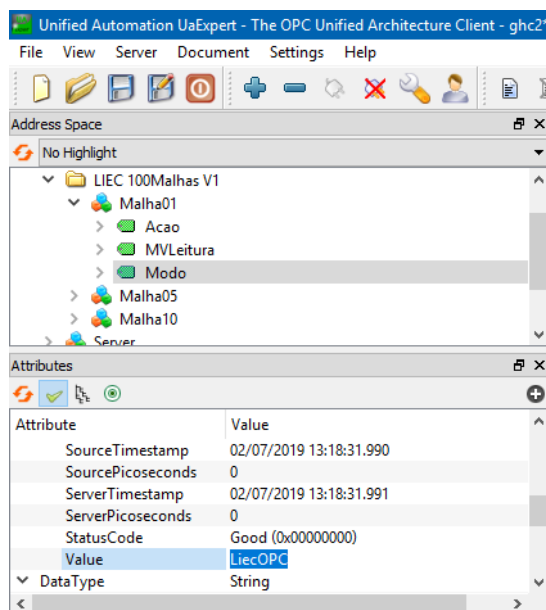


Fonte: Próprio Autor.

Posteriormente foi realizado o processo inverso. Primeiro foi realizada uma operação de escrita no ambiente UA (Figura 33) e depois foi lido o valor correspondente no ambiente DA como mostra a Figura 34.

Para os testes realizados foi verificada a capacidade da aplicação de atualizar o valor de uma variável sempre que o item correspondente sofre uma alteração no seu valor e vice-versa.

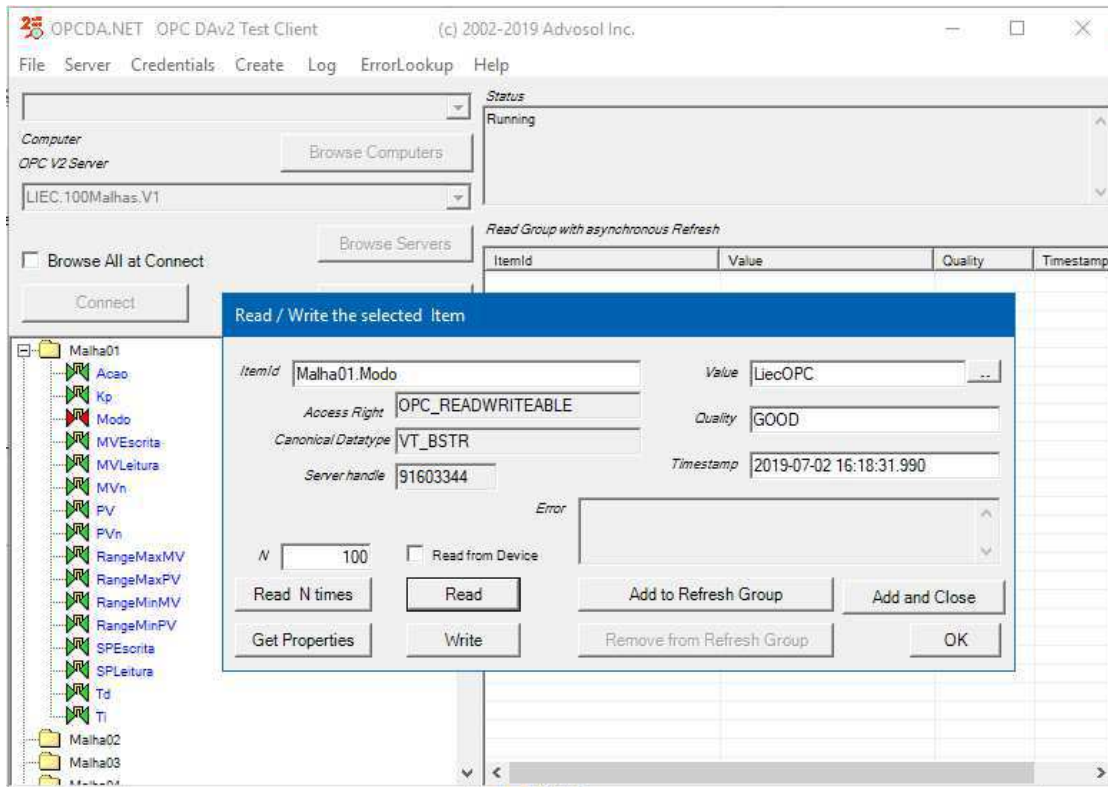
Figura 33 – Alteração no valor de uma variável servidor UA.



Fonte: Próprio Autor.

Durante o desenvolvimento da aplicação foi verificado que essa correspondência de valores pode ser prejudicada caso haja, por algum motivo, uma inconsistência de tipo de valor entre os itens e suas variáveis correspondentes. Por isso, é preciso garantir que, ao criar o servidor OPC UA, os tipos das variáveis sejam compatíveis com os tipos dos seus respectivos itens.

Figura 34 – Leitura do valor do item DA correspondente.



Fonte: Próprio Autor.



## 5 | Conclusões

Diante da proposta deste trabalho, foi alcançado o objetivo de possibilitar o intercâmbio de informações entre servidores OPC DA e clientes OPC UA e permitir, assim, que usuários de produtos baseados no padrão OPC Clássico possam ter acesso a alguns recursos próprios do OPC *Unified Architecture*.

Do ponto de vista de desenvolvimento do projeto, é importante salientar que sua realização se deve em grande parte à utilização de SDKs específicos para cada padrão OPC, bem como o uso do ambiente *Visual Studio* e da linguagem C#, que facilitaram e agilizaram as implementações em código mediante as funções e recursos oferecidos.

Por fim, considera-se que o caminho trilhado para o desenvolvimento desta aplicação pode servir de base para futuras implementações nas quais pode se dar foco a parâmetros como eficiência e robustez a fim de promover o aprimoramento da solução apresentada.



# Referências

- [1] MAHNKE, Wolfgang; LEITNER, Stefan-Helmut; DAMM, Matthias. *OPC Unified Architecture*. Springer Science & Business Media, 2009. Citado 9 vezes nas páginas 1, 3, 4, 5, 6, 7, 8, 9 e 10.
- [2] GONÇALVES, Robson Neves. *Desenvolvimento de Servidores OPC DA, OPC UA e Wrappers para aplicação em Automação*. 2012. Citado na página 1.
- [3] OPC FOUNDATION. *OPC Unified Architecture Part 1 :Overview and Concepts*. Release 1.04. 22 de novembro de 2017. Citado na página 7.
- [4] OPC FOUNDATION. *OPC Unified Architecture Part 3: Address Space Model*. Release 1.04. 22 de novembro de 2017. Citado 2 vezes nas páginas 8 e 9.
- [5] OPC FOUNDATION. *OPC Unified Architecture Part 8: Data Access*. Release 1.04. 01 de novembro de 2017. Citado na página 10.
- [6] ADVOSOL. *OPCDA.NET Client Component*. Reference Manual. Disponível em: <<https://advosol.com/Manuals/OpcDaNet/webframe.html>>. Acesso em: 8 abr. 2019. Citado na página 11.
- [7] UNIFIED AUTOMATION. *.NET Based OPC UA Client/Server SDK Documentation*. Release 2.6.1. 30 de Outubro de 2018. Citado na página 11.