UNIVERSIDADE FEDERAL DE CAMPINA GRANDE
CENTRO DE ENGENHARIA ELÉTRICA E INFORMÁTICA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

José Adeilmo Nunes Barbosa Junior

Relatório de Estágio Supervisionado

Laboratório de Instrumentação Eletrônica e Controle

Campina Grande - PB

José Adeilmo Nunes Barbosa Junior

Relatório de Estágio Supervisionado

Relatório de Estágio Supervisionado submetido à Coordenação de Graduação em Engenharia Elétrica da Universidade Federal de Campina Grande, Campus Campina Grande, como parte dos requisitos necessários para obtenção do título de Graduado em Engenharia Elétrica.

Universidade Federal de Campina Grande - UFCG

Centro de Engenharia Elétrica e Informática - CEEI

Departamento de Engenharia Elétrica - DEE

Orientador: Rafael Bezerra Correia Lima, D.Sc.

Campina Grande, Brasil

29 de novembro de 2019

José Adeilmo Nunes Barbosa Junior

Relatório de Estágio Supervisionado

Relatório de Estágio Supervisionado submetido à Coordenação de Graduação em Engenharia Elétrica da Universidade Federal de Campina Grande, Campus Campina Grande, como parte dos requisitos necessários para obtenção do título de Graduado em Engenharia Elétrica.

Trabalho aprovado em: / /

Rafael Bezerra Correia Lima, D.Sc.
Orientador

George Acioli Júnior, D.Sc.
Convidado

Campina Grande, Brasil
29 de novembro de 2019

Dedico este trabalho a Edna Gomes de Medeiros, amada mãe e a José Adeilmo Nunes Barbosa, amado pai

Agradecimentos

A minha mãe Edna por estar do meu lado em todos os momentos, meu porto seguro, acho que nunca conseguirei demonstrar o quando sua presença é crucial em minha vida. A meu pai Adeilmo por seu apoio durante toda universidade e me permitir conhecê-lo melhor. A meus irmãos, Ananda Sophia e Erick Lohan por serem tão especias em minha vida.

Aos meus familiares que sempre me apoiaram, principalmente meus avós maternos Everaldo e Marlene e minha avó paterna Irami, por seu carinho especial e me fazer sentir a pessoa mais sortuda do mundo. A Ellen minha querida e amada namorada por ter me suportado durante todo o tempo da universidade, mesmo nos famosos finais de período, sempre me ajudando a resolver problemas externos e internos. A Ravi, meu irmão/primo/amigo/filho, agradeço por todos conhecimentos compartilhados úteis e principalmente os inúteis. A Karol, amiga de longas datas por seus conselhos e companheirismo.

A membros da família Ferreira que me aceitaram em sua família de braços abertos em especial a dona Eunice Serafim e o senhor José Ferreira.

Aos meus amigos do grupo Armário que me adotaram e fizeram todos os dias na universidade serem toleráveis. Agradeço ao restaurante universitário (RU) por sua enorme fila onde os momentos com meus amigos foram prolongados, as noites de pizza, *just dance*, aos piquenique, sou grato a todos os momentos que passei com vocês e espero que continuemos nos encontrando.

Ao Laboratório de Instrumentação Eletrônica e Controle (LIEC) ou simplesmente casa, sou grato ao professor Péricles Barros por ter me convidado para fazer parte desta enorme família e por sempre ter acreditado no meu potencial, ao professor Rafael Bezerra, meu orientador e exemplo, ao Professor George Acióli pelos seus conselhos. Aos meus companheiros de laboratório, Breno Sant'Anna, Egydio, Lucas Porto e Paulo Roberto pelos momentos de procrastinação e diversão no laboratório. Em especial ao meu grande amigo Luquinhas que se tornou um dos amigos mais queridos, mesmo quando me expulsou de sua mesa, obrigado por tudo.

Por fim agradeço a eu mesmo, por ter seguido em frente e dado o melhor de si independente das circunstancias, sempre me dando a certeza de que fiz o melhor que pude em todos os momentos.

"Were you born to resist or be abused?
I swear I'll never give in
I refuse
Is someone getting the best, the best, the best, the best of you?"
(Foo Fighters - *Best Of You*)

Resumo

Este relatório apresenta as atividades realizadas pelo aluno José Adeilmo Nunes Barbosa Junior durante o Estágio Supervisionado no Laboratório de Instrumentação Eletrônica e Controle (LIEC), pertencente ao Departamento de Engenharia Elétrica (DEE) da Universidade Federal de Campina Grande (UFCG), sob orientação do professor Rafael Bezerra Correia Lima e supervisão do Professor George Acioli Júnior. O principal objetivo deste trabalho é a criação de uma interface OPC UA para o sistema de inspeção do laboratório, abordando temas recorrentes na graduação, como desenvolvimento de software e padrão OPC UA. Além disso, estudos associados à manutenção preditiva foram realizados.

Palavras-chaves: OPC UA; Manutenção Preditiva; Sistema de inspeção.

Abstract

This report presents the activities performed by the student José Adeilmo Nunes Barbosa Junior during the Supervised Internship at the Laboratory of Electronic Instrumentation and Control (LIEC), belonging to the Department of Electrical Engineering (DEE) of the Federal University of Campina Grande (UFCG), under the guidance of the Professor Rafael Bezerra Correia Lima and supervision of Professor George Acioli Júnior. The main objective of this apprenticeship is to create an OPC UA interface for the lab inspection system, addressing recurring graduation topics such as software development and OPC UA standard. In addition, studies associated with predictive maintenance were performed.

Key-words: OPC UA; Predictive Maintenance; Inspection System.

Lista de ilustrações

Lista de abreviaturas e siglas

ABNT	Associação Brasileira de Normas Técnicas
COM	*Component Object Model*
DA	*Data Acess*
DCOM	*Distributed Component Object Model*
DEE	Departamento de Engenharia Elétrica
LIEC	Laboratório de Instrumentação Eletrônica e Controle
MVP	*Model-View-Presenter*
MVVM	*Model-View-ViewModel*
PDA	*Personal digital assistant*
OPC	*OPE for Process Control*
SDK	*Software development kit*
TCP	*Transmission Control Protocol*
UA	*Unified Architecture*
UFCG	Universidade Federal de Campina Grande

Sumário

1 Introdução

O estágio supervisionado, cujas atividades são descritas neste relatório, teve duração de 360 horas (12 créditos) e foi realizado no Laboratório de Instrumentação Eletrônica e Controle (LIEC), durante o período de 12 de agosto de 2019 até 26 de novembro de 2019, sob a supervisão do professor e engenheiro eletricista George Acióli Júnior e orientação do professor Rafael Bezerra Correia Lima.

O estágio supervisionado tem como objetivo o cumprimento das exigências da disciplina Estágio Curricular, integrante da grade curricular do Curso de Engenharia Elétrica da Universidade Federal de Campina Grande. Essa disciplina é indispensável para a formação profissional, visto que consolida os conhecimentos adquiridos durante o curso de forma prática.

O estágio teve como principal objetivo a criação de uma interface OPC UA que fosse independente de plataforma para um sistema de inspeção já existente no laboratório, colocando em prática conhecimentos de diversas disciplinas cursadas durante o curso como:

- Sistemas de automação industrial;
- Informática industrial;
- Técnicas de programação.

Desenvolvendo atividades de:

1. Revisão bibliográfica de OPC DA e OPC UA;
2. Revisão bibliográfica de desenvolvimento de softwares;
3. Levantamento de requisitos da interface a ser desenvolvida;
4. Desenvolvimento de um aplicativo para gerenciamento;
5. Desenvolvimento de um aplicativo para inspeção;
6. Realização da documentação.

2 Local do estágio

Localizado na Universidade Federal de Campina Grande (UFCG) no campus de Campina Grande, o Laboratório de Instrumentação Eletrônica e Controle (LIEC) é um laboratório pertencente ao Departamento de Engenharia Elétrica (DEE). Integrado por professores doutores, alunos de pós-graduação e de graduação, esse laboratório tem como principal objetivo desenvolver atividades e projetos ligados as áreas de automação, controle e instrumentação.

Figura 1 – Fachada do LIEC.

Fonte: autoria própria.

Com uma área de aproximadamente $600m^2$, o LIEC que pode ver visto na figura 1 conta com oito laboratórios de desenvolvimento, duas salas de apoio técnico, sala para apresentação de trabalhos, salas para pós-graduação e professores. Dentre as principais atividades desenvolvidas no laboratório, pode-se destacar as seguintes:

- Laboratório de Aplicações Wireless - desenvolvimento de soluções baseadas em dispositivos móveis para ambientes industriais;
- Laboratório de Automação Industrial - sintonia de controladores PID industriais (mono e multivariável), automação industrial, instrumentação industrial, IHM industrial, avaliação de confiabilidade em malhas de controle;

- Laboratório de Controle e Otimização - projeto e sintonia de PID, modelagem e simulação de processos, sistemas supervisórios;
- Laboratório de Instrumentação Eletrônica - projeto e sintonia de PID;
- Laboratório de Redes Industriais - estudos de técnicas e tecnologias para a comunicação entre dispositivos industriais;
- Laboratório de RFID - desenvolvimento de aplicações baseadas em tecnologia RFID para ambientes industriais;
- Laboratório de Ultrassom - desenvolvimento de sensor de incrustação, desenvolvimento de técnicas de medição de incrustação.

No LIEC, alunos de pós-graduação e de graduação encontram um ambiente propício ao aperfeiçoamento dos conhecimentos teóricos e das habilidades práticas, por intermédio das diversas pesquisas e outras atividades realizadas no mesmo.

3 Fundamentação Teórica

3.1 Manutenção

A manutenção é definida pela norma ABNT NBR 5462/1994 como a combinação de todas as ações técnicas e administrativas, incluindo as de supervisão, destinadas a manter ou recolocar um item em um estado no qual possa desempenhar uma função requerida.

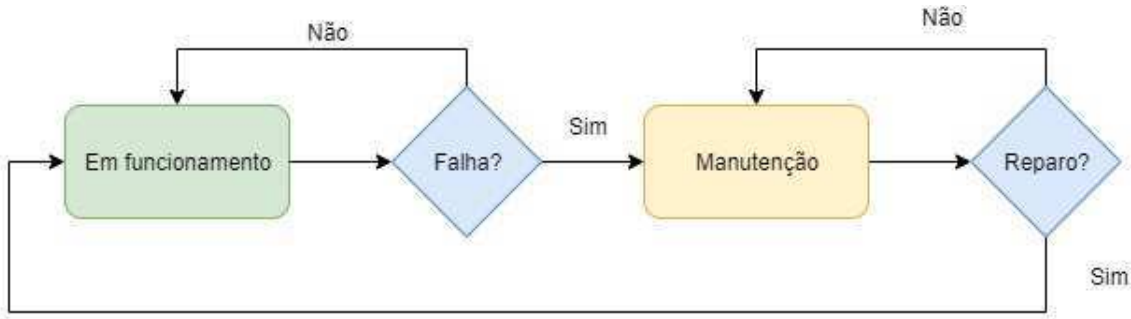
Esta atividade se torna crítica no ambiente industrial, pois a falta de uma correta manutenção pode significar um prejuízo: Financeiro, parada indesejada da linha de produção; Ecológico, vazamento de óleo; Humano, ocasionamento de um acidente de trabalho.

Dado a sua importância foram desenvolvidas diversas políticas de manutenção, sendo as manutenções corretiva, preventiva e preditiva as formas mais difundidas.

3.1.1 Manutenção corretiva

A manutenção corretiva na norma ABNT NBR 5462/1994 é definida como manutenção efetuada após a ocorrência de uma pane destinada a recolocar um item em condições de executar uma função requerida. Em outras palavras, a manutenção ocorrerá quando o equipamento apresentar algum defeito que afete o seu funcionamento como podemos observar na figura 2.

Figura 2 – Fluxograma da manutenção corretiva.



Fonte: autoria própria.

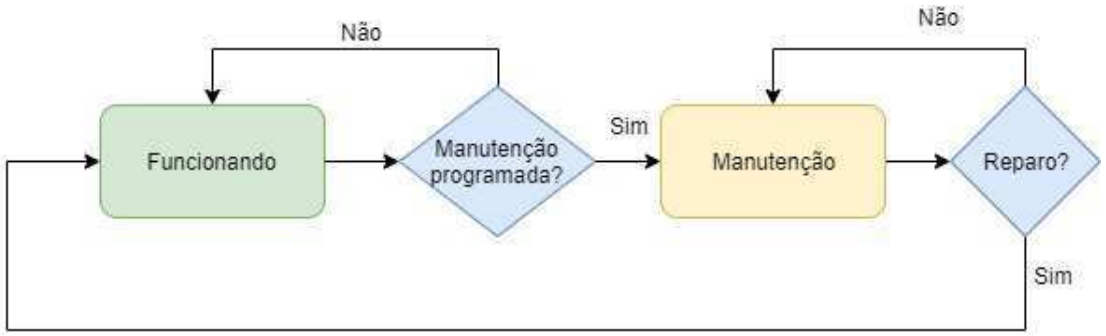
3.1.2 Manutenção preventiva

Manutenção preventiva é definida pela norma ABNT NBR 5462/1994 como manutenção efetuada em intervalos predeterminados, ou de acordo com critérios prescritos,

destinada a reduzir a probabilidade de falha ou a degradação do funcionamento de um item.

É o mesmo tipo de manutenção que é utilizado em diversos sistemas dos automóveis como troca de óleo, troca de correia dentada, entre outros. São definidos intervalos periódicos que podem ser intervalos de tempo de uso ou qualquer outro parâmetro que represente a degradação do equipamento e ao chegar ao fim do período definido é realizada a manutenção como podemos observar na figura 3.

Figura 3 – Fluxograma ideal da manutenção preventiva.



Fonte: autoria própria.

Este tipo de manutenção previne a falha do equipamento podendo ser gerenciado com um sistema de controle da manutenção relativamente simples, sendo necessário registrar o dia que a manutenção foi realizada e quando será realizada a próxima manutenção.

Os principais problemas desta política são as paradas programadas da linha produtiva de forma periódica mesmo que não haja necessidade, além de eventuais substituições de peças que ainda possuem tempo de vida útil, gerando um gasto desnecessário com peças de reposição e ocasionando limitação da produção pela diminuição da disponibilidade do equipamento.

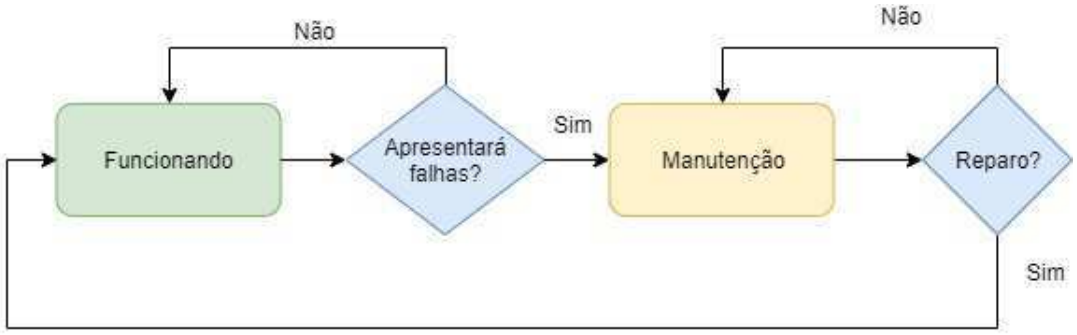
3.1.3 Manutenção preditiva

A manutenção preditiva é o tipo de manutenção que visa maximizar o tempo de disponibilidade dos equipamentos. É definida na norma ABNT NBR 5462/1994 como manutenção que permite garantir uma qualidade de serviço desejada, com base na aplicação sistemática de técnicas de análise, utilizando-se de meios de supervisão centralizados ou de amostragem, para reduzir ao mínimo a manutenção preventiva e diminuir a manutenção corretiva.

Este tipo de manutenção consiste em utilizar variáveis físicas para avaliar se o equipamento está prestes a apresentar alguma falha. A principal diferença entre a ma-

nutenção preditiva para a preventiva é no motivo de realização da manutenção, como podemos observar na figura 4.

Figura 4 – Fluxograma ideal da manutenção preditiva.



Fonte: autoria própria.

Desta forma as interrupções da linha de produção são realizadas na iminência da falha e a substituição da peça é realizada quando realmente é necessária. Aprimorando assim a manutenção preventiva, porém para este tipo de manutenção ser eficiente é necessário um sistema de gerenciamento da manutenção com auxílio de inspeções periódicas, pois a não realização da inspeção no prazo pode significar na falha não prevista do equipamento, ocasionando a utilização da manutenção corretiva neste.

Para otimizar as inspeções, empresas tem investido em gerenciadores para esta tarefa, que são softwares que auxiliam no acompanhamento das atividades a serem realizadas e na distribuição das tarefas de inspeção. Existem sistemas limitados onde ordens de inspeções são impressas e preenchidas à mão pelos funcionários responsáveis e após a realização da inspeção deve-se digitalizar as informações, também existem sistemas conectados onde as inspeções são realizadas com o auxílio de um dispositivo móvel (PDA) e os dados são direcionados automaticamente para o banco de dados.

3.2 OPC

A comunicação entre dispositivos de diferentes fabricantes era um empecilho no desenvolvimento e implementação de redes industriais. Uma empresa ao escolher um determinado produto, deveria portanto adotar todos os dispositivos necessários desta mesma marca de modo a tornar possível a comunicação entre estes (IWANITZ; LANGE, 2001). A comunicação com dispositivos de marcas diferentes necessitavam de tradutores (*gateways*), à medida que a aplicação se tornava complexa o número de *gateways* crescia e tornava a rede do chão de fábrica mais caótica.

Em 1995 diversos fabricantes de dispositivos para automação, Fisher-Rosemount,

Rockwell Software entre outras reuniram se para tentar criar um padrão de acesso a dados utilizando a tecnologia COM/DCOM da Microsoft, o OPC.

A sigla OPC significa OLE (Microsoft *Object Linking & Embedding*) *for Process Control*, ou seja, é a utilização do conceito da ferramenta de vinculação e incorporação de objetos da Microsoft nos sistemas de controle de processos.

Apenas em 1996 o OPC teve sua primeira especificação publicada OPC *Data Access* ou OPC DA, no mesmo ano que foi publicado, diversos fabricantes começaram a utilizá-lo em seus produtos. A sua consolidação foi marcada pela criação da OPC Foundation, organização formada por diversas empresas da área. A OPC Foundation tem como objetivo criação das especificações, validar e certificar a implementações das especificações.

Após a publicação da especificação do OPC DA, seguindo a mesma tecnologia da Microsoft foram criadas as especificações OPC EA e OPC HDA, porém foram pouco utilizadas. Foi surgindo a necessidade de que o OPC não dependesse mais da plataforma da Microsoft e uma nova abordagem para a modelagem dos dados para tornar o padrão mais flexível.

Para solucionar os problemas encontrados no OPC DA, em 2003 foi lançado 13 especificações do OPC *Unified Architecture*, ou OPC UA e a partir deste momento o padrão foi dividido em OPC Clássico o padrão dependente da plataforma Microsoft e OPC UA as especificações independente de plataforma.

3.2.1 OPC Clássico

O OPC Clássico teve seu início com a publicação da especificação OPC *Data Access* (OPC DA), seguido das especificações OPC *Alarms Events* (OPC AE) e OPC *Historical Data Access* (OPC HDA). Todas estas especificações utilizavam a tecnologia COM/DCOM de propriedade da Microsoft.

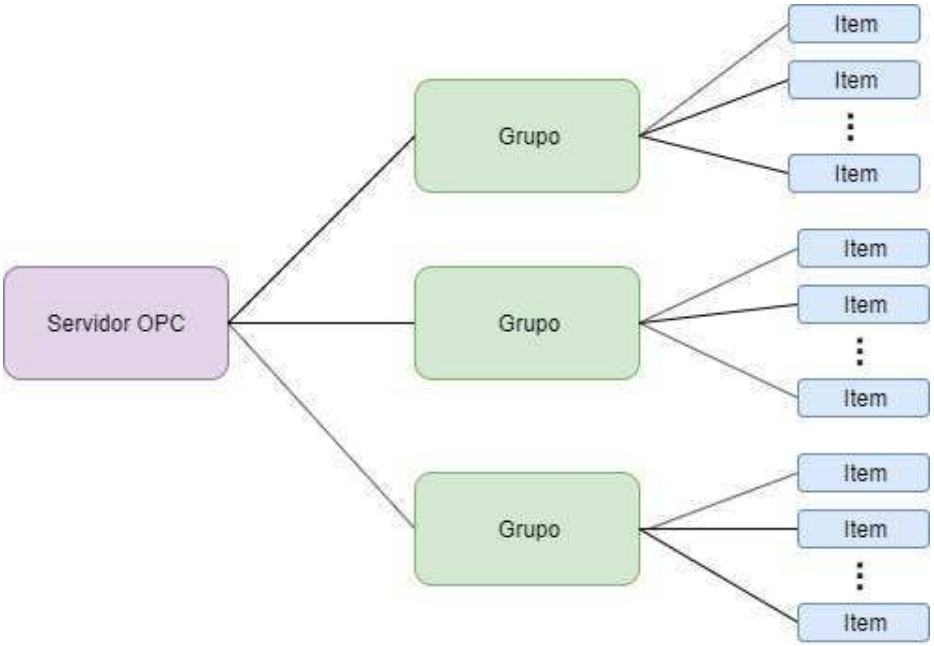
O OPC DA é a interface que permite a leitura, escrita e monitoramento de variáveis do sistema, é possível o monitoramento de uma variável de um CLP em uma IHM, viabilizando a modificação da mesma via IHM ou por meio de clientes OPC DA em *desktops*.

Podemos observar na figura 5, que na arquitetura do OPC DA existem 3 diferentes tipos de objetos, são estes:

- Servidor: onde são executadas as interfaces entre as aplicações;
- Grupo: fica em uma camada superior, é onde os itens são organizados e onde ocorre o controle de atualização dos valores;

- Item: representa uma variável especifica do sistema, além do valor da variável, possui informações sobre a qualidade da informação;

Figura 5 – Arquitetura do OPC DA.



Fonte: autoria própria.

O OPC AE é a interface que permite a utilização de alarmes e eventos que podem ser programados para informar o cliente quando um determinado estado é alcançado, gerando um evento para ser utilizado em outra parte do sistema ou enviando um alarme para os clientes.

Por fim, temos o OPC HDA que é a interface que permite o acesso aos dados históricos do sistema, ele possui 3 formas principais de requisição:

- Na primeira forma o cliente solicita o histórico de uma ou mais variáveis a partir de uma data e a quantidade máxima de valores desejados e o servidor retorna os valores armazenados desta data em diante respeitando o limite de valores a serem lidos definidos pelo cliente;
- Na segunda forma a diferença é que o cliente define uma ou mais variáveis e informa a janela de tempo que ele deseja os dados e o servidor por sua vez retorna todos os valores existentes nesta janela;
- Na terceira forma o servidor agrega valor aos dados, adicionando informações como *timestamp* e qualidade do valor, informando se o valor da leitura tem boa qualidade, qualidade incerta ou má qualidade.

Na indústria estes sistemas foram largamente utilizados, principalmente o OPC DA que em 2009 chegou a ser implementado em 99% dos produtos que utilizavam a tecnologia OPC (MAHNKE; LEITNER; DAMM, 2009). Porém alguns problemas foram identificados nas especificações existentes e notou-se a necessidade de desenvolver uma nova especificação, entres os problemas estavam a dependência da plataforma Microsoft e limitação para modelar sistemas complexos, seus valores não continham o significado físico ou lógico, então surgiu o OPC *Unified Architecture*.

3.2.2 OPC *Unified Architecture*

Como o nome já sugere o OPC UA tem como uma das principais mudanças em relação ao OPC Clássico, a independência de plataforma. Outro grande avanço do OPC UA é sua modelagem, onde são utilizados conceitos de programação orientada a objeto para abstração dos sistemas reais. Em OPC DA um item possuía apenas seu valor, um objeto em OPC UA possui um significado associado, além dos avanços na modelagem dos dados o OPC UA dá suporte funcionalidades existentes nas especificações OPC AE e OPC HDA.

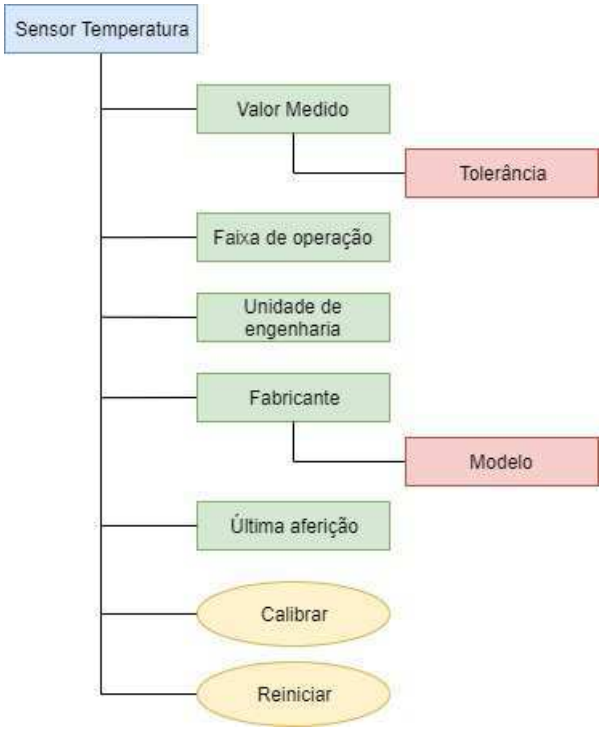
Para exemplificar o poder desta modelagem, vamos modelar um sensor de temperatura. Para o OPC DA o sensor de temperatura possui apenas o valor de temperatura medido, já para o OPC UA além desta informação podemos agregar ao sensor sua faixa de operação, a unidade de engenharia associada ($^{\circ}C$, $^{\circ}F$ ou K), a marca do fabricante, modelo do sensor, tolerância e quando foi realizada a última aferição como podemos observar na figura 6, além de ser possível criar métodos para calibração e reiniciar o sensor.

Para superar a dependência do protocolo de comunicação COM/DCOM foi utilizado na camada de transporte uma versão otimizada do TCP, por sua performance na rede industrial. Foi realizado uma adaptação para versão do TCP já utilizada em aplicações web.

Atualmente o OPC UA possui 14 especificações representadas na figura 7, podemos classificá-las em 3 grupos (MAHNKE; LEITNER; DAMM, 2009):

- Funcionamento do OPC UA: neste grupo estão as especificações que definem como o OPC UA é formado;
- Formas de acesso ao servidor: aqui estão as especificações referentes a como o servidor pode ser utilizado;
- Servidor de descoberta: É a especificação que descreve o servidor de descoberta, ele tem a função de auxiliar os clientes a encontrarem os servidores disponíveis na rede.

Figura 6 – Exemplo de um sensor de temperatura no OPC UA.



Fonte: autoria própria.

Figura 7 – Especificações existentes do OPC UA.

Fonte: Adaptado de (MAHNKE; LEITNER; DAMM, 2009)

A base para a modelagem em OPC UA são nós e referências, mesmo possuindo apenas 2 tipos de variáveis há uma grande flexibilidade ao se modelar, de forma simplificada tempos que:

- **Nós**: Possuem diferentes tipos de nós, com diferentes significados. Eles representam as instâncias ou definições dos tipos, todos nós possuem alguns atributos em comum como por exemplo: *NodeId*, identificador único do nó; *Description*, breve descrição do nó, *NodeClass*, qual o tipo do nó. As operações de leitura, escrita e busca são voltadas para os nós.
- **Referências**: As referências conectam dois nós, estabelecendo alguma relação entre eles, assim como nós, existem diferentes tipos de referências. As referências não possuem atributos, elas dão semântica a modelagem. Na figura 6 temos um exemplo de referências do tipo *HasComponet* liga dois nós informando que um nó possui o outro, na modelagem é representando uma propriedade, todas as linhas conectando os retângulos verdes, que são nós, conectados ao nó do sensor são referências do tipo *HasComponet*.

Além de modelagem de dados o OPC UA suporta a criação de métodos que podem ser chamados pelos clientes. Os métodos são semelhantes a métodos utilizados normalmente na programação, podem necessitar ou não de argumentos de entrada, é executado um bloco de código e ao final retornam se o método foi executado com sucesso e podem possuir argumentos de saída para o cliente.

Um exemplo de método que pode ser criado é o método para partida de uma máquina qualquer, o cliente chama o método do objeto desejado, o servidor executa o passo a passo para partir o equipamento que pode ser traduzido como escrever algo em alguns nós e monitorar outros nós e ao fim retorna ao cliente que o método foi executado com sucesso que significa nesse caso que a máquina foi partida com sucesso.

3.2.3 SDK OPC Foundation

Software development kit (SDK) é uma biblioteca de classes que auxiliam no desenvolvimento de aplicações, no caso para a implementação de aplicações OPC UA sem a existência de SDK's seria necessário implementar todo o padrão de comunicação desde sua camada de transporte encapsulando os dados para transmissão, segurança até os métodos de interação com o servidor para o caso de uma aplicação OPC UA cliente. Com os SDK's podemos abstrair todas as particularidades de como o padrão é implementado e apenas utilizamos os métodos de alto nível como: conexão, leitura de dados, escrita de dados, executar um método do servidor.

Diversas empresas como a Unified Automation, Advosol e Softing vendem estes SDK para implementação de clientes e servidores OPC UA, cada um possui sua particularidade em como são usados os seus métodos, porém todos implementam o padrão, por ser voltado para aplicações industriais os preços costumam ser altos. A OPC Foundation disponibiliza em seu repositório público de forma gratuita o seu SDK.

3.3 Engenharia de software

Criada para tentar contornar a crise do software dos anos 70, a engenharia de software começou a tratar o desenvolvimento de software com uma visão de engenharia (sistemático, controlado e de qualidade mensurável), criando ferramentas e boas práticas para auxiliar na criação de sistemas complexos (MCCONNELL, 2004).

No desenvolvimento duas áreas da engenharia de software foram adotadas, são elas: Controle de versão e Padrões de arquitetura de software.

3.3.1 Controle de versão

O controle de versão é um sistema que registra as mudanças feitas em um arquivo ou um conjunto de arquivos ao longo do tempo de forma há ser possível recuperar versões específicas. Ao utilizá-lo no desenvolvimento de aplicações auxilia no controle do que está sendo feito, caso uma alteração do código faça com que uma funcionalidade já existente deixe de funcionar é possível desfazer as mudanças (COLLINS-SUSSMAN; FITZPATRICK; PILATO, 2004).

Utilizando esta ferramenta auxilia na união do código de programas implementados por mais de uma pessoa, onde define-se em que parte do projeto cada programador irá modificar e ao enviar suas alterações para o controle ele realiza a união do código.

3.3.2 Padrões de arquitetura de software

Existem diversas boas práticas que desenvolvedores deveriam seguir, os padrões de arquitetura são uma delas. Os padrões de arquitetura são formas de organizar a sua aplicação de forma que fique modularizada e organizada, o que facilita no entendimento do seu software, correção de bugs e aumenta a possibilidade de reutilização de código, pois alguns de seus métodos são independentes de sua aplicação atual, caso seja necessário em outro programa basta replicar o que já foi feito.

Existem diversos padrões de arquitetura para as mais diversas linguagens, cada um possui um nível de desacoplamento diferente ou apenas divide a aplicação em módulos diferentes.

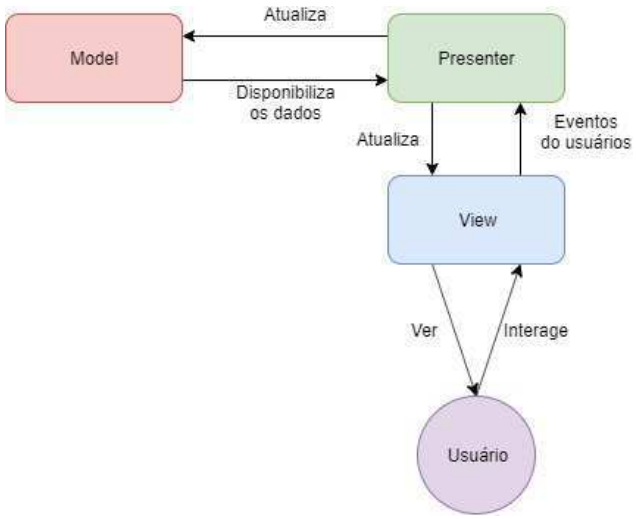
3.3.2.1 MVP

O MVP (*Movel-View-Presenter*) possui 3 entidades:

- *Model*: Parte do código que possui a definição das classes a serem utilizadas, são os modelos da sua aplicação;
- *View*: É a interface gráfica que o usuário irá visualizar e utilizar;
- *Presenter*: Ele é responsável por lidar com os eventos gerados pelo usuário, atualizar a interface do usuário e possui a lógica da aplicação.

Na figura 25 podemos observar melhor o funcionamento das partes do MVP. O usuário irá visualizar as informações expostas na *view* e irá interagir com a mesma por exemplo clicando em algum botão que lá esteja, a *view* acionará um evento do *presenter* que o usuário clicou no botão, o *presenter* por sua vez irá executar o que foi programado podendo atualizar os dados dos modelos, atualizar as informações mostradas ao usuário ou até mesmo mudando a tela que está sendo apresentada, o modelo está encarregado de armazenar os dados e disponibilizá-lo ao *presenter*.

Figura 8 – Diagrama do padrão MVP.



Fonte: autoria própria.

3.3.2.2 MVVM

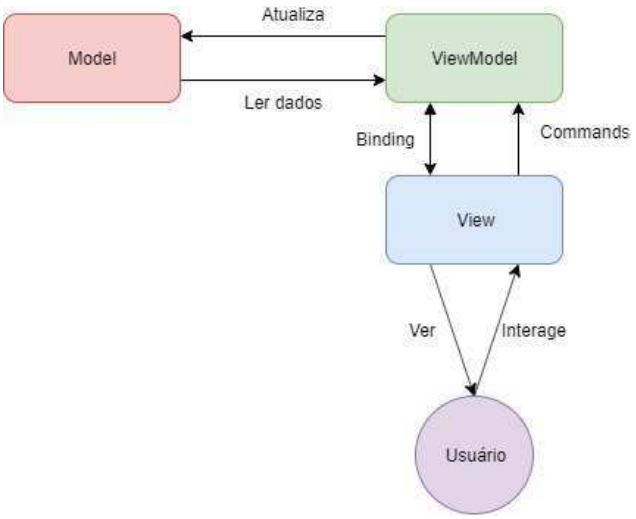
O MVVM (*Model-View-ViewModel*) assim como o MVP possui 3 divisões:

- *Model*: Assim como o MVP, são os modelos de nossa aplicação;

- *View*: Igual ao MVP, são as telas da nossa aplicação, o que o usuário visualiza e interage;
- *ViewModel*: É semelhante ao *presenter*, porém utiliza uma abordagem diferente para atualizar a *view*. Utiliza de um mecanismo denominado de *binding* que liga suas propriedades a elementos das *view*.

O MVVM utiliza notificações, quando o usuário interage com a *View* o evento sinaliza para o *ViewModel* que por sua vez executa o que deve, atualizando ou não os dados dos *Models* ou alterando a interface. A atualização da *View* é realizada por meio de *commands* que são notificadores, os objetos da interface são ligados *binding* a propriedades do *ViewModel* e quando estas propriedades são alteradas o *ViewModel* notifica a *View* para ela atualizar o seu valor. Como podemos ver, existe uma dependência entre a *View* e o *ViewModel* algo que não existia no MVP porém em aplicações que podem utilizar o XAML este tipo de padrão de arquitetura funciona bem e consegue manter uma boa organização do código.

Figura 9 – Diagrama do padrão MVVM.



Fonte: autoria própria.

3.4 Plataformas

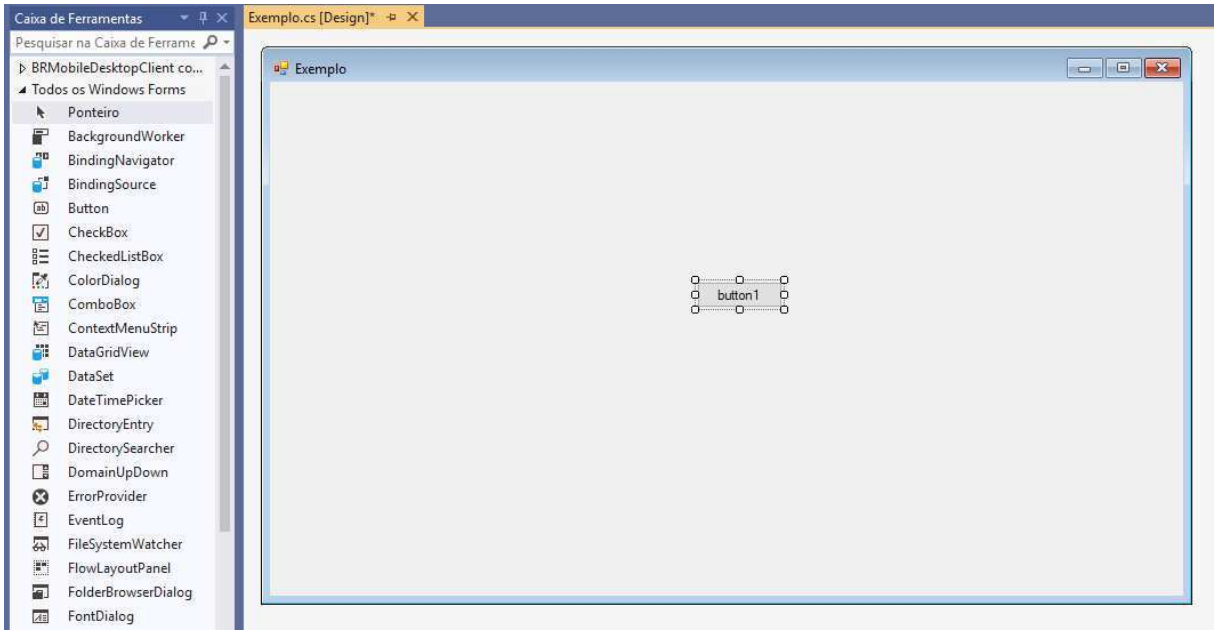
Uma das primeiras etapas em um desenvolvimento de software é a escolha da plataforma alvo, onde nossa aplicação será utilizada. A Microsoft possui diversas opções para desenvolvimento de software entre elas o Windows Forms e o Xamarin Forms, ambos utilizam o C# como linguagem base. Outro ponto importante é verificar qual o alvo do SDK que será utilizado, onde .NET .Framework é suportado apenas em aplicações Windows, já .NET Standard possui suporte em diversas plataformas.

3.4.1 Windows Forms

O Windows Forms é uma biblioteca de classes voltada para o desenvolvimento de interfaces gráficas para aplicações Windows, ela contém definições de classes que representam os itens de controle da interface gráfica como: botão, etiqueta e campo de texto definições janela da aplicação chamado de *forms* entre outros.

Utilizando o Windows Forms no ambiente de desenvolvimento integrado (IDE) do Visual Studio ®, interfaces gráficas são criadas por meio de uma ferramenta de *drag and drop*, ou seja, basta selecionar objeto desejado e soltá-lo na posição desejada, na figura 10 temos à esquerda disponíveis todos os elementos suportados pelo Windows Forms e à direita a visualização gráfica da tela criada, para adicionar um novo elemento na tela basta escolher na lista de elementos e posicioná-lo na tela. Todo o código responsável pela instanciação dos elementos e posicionamento é gerado automaticamente pela ferramenta, sendo assim o programador pode focar apenas na lógica da aplicação, otimizando o tempo de desenvolvimento.

Figura 10 – Parte da tela de criação de *views* do WinForms.



Fonte: autoria própria.

Além da facilidade para desenvolvimento de interfaces gráficas o Windows Forms possui vasta documentação para auxiliar na sua aprendizagem. São disponibilizados diversos guias introdutórios explicando como criar aplicações simples utilizando-o. O padrão de arquitetura de software usuais para o Windows Forms é o *Model-View-Control* (MVC) e o *Model-View-Presenter* (MVP)

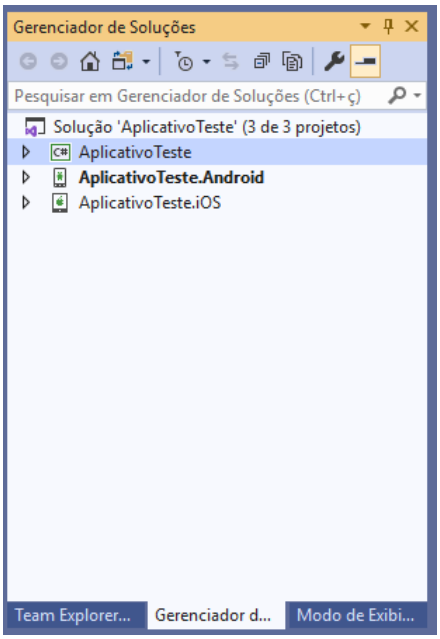
3.4.2 Xamarin Forms

Também da Microsft o Xamarin Forms é uma interface de programação de aplicações (API) para criação de aplicações multi-plataforma (*cross-plataform*), com ela é possível criar aplicativos android, iOS e UWP (Windows) que possuam código base, quase que em sua totalidade, compartilhado.

O Xamarin Forms não possui um design igual o Windows Forms para auxiliar na criação de telas, é utilizado uma linguagem de marcação XAML, semelhante a XML, para este propósito. Também tem-se a opção utilizar o próprio C# para a criação das telas das aplicações, porém geralmente tornam o código confuso, o XAML simplifica a criação das interfaces gráficas e na modularização do código.

Ao cria um projeto em Xamarin Forms são criados, um projeto que possui a parte compartilhada da aplicação, códigos base e telas e um projeto para cada plataforma alvo. Na figura 11 vemos o projeto compartilhado "AplicativoTest", onde ficará nosso código compartilhado, logo após temos o projeto tendo como plataforma alvo o Android "AplicativoTest.Android"caso seja necessário ter acesso a funções especificas do android, adição de imagens entre outros iremos recorrer a este projeto, por fim, temos o projeto para iOS "AplicativoTest.iOS"iremos utilizá-lo nas mesmas condições que utilizamos o projeto Android.

Figura 11 – Projetos criados em uma aplicação Xamarin Forms para Android e iOS.



Fonte: autoria própria.

Pelo fato de que utilizamos o XAML para programar a interface gáfica do Xamarin Forms o padrão de arquitetura de software que é indicada pela Microsoft é o MVVM.

4 Atividades Realizadas

Na presente seção serão apresentas as principais atividades desenvolvidas pelo estagiário no Laboratório de Instrumentação Eletrônica e Controle (LIEC). O estágio supervisionado foi realizado em Campina Grande - Paraíba, na sala de IoT Industrial deste laboratório.

Na seção 4.1 discorre-se sobre a atividade de entendimento do sistema de monitoramento baseado em OPC UA desenvolvido pelo laboratório.

Na seção 4.2 é apresentado o SDK escolhido e o método utilizado para aprende-lo.

Na seção 4.3 discorre-se as etapas de criação da interface OPC UA utilizando Windows Forms.

Na seção 4.4 é apresentado a aplicação Xamarin Forms criada para realização das inspeções.

4.1 LIEC-Inspections

A primeira atividade exercida durante o estágio foi estudar o sistema de inspeção já existente no laboratório. O LIEC-Inspections é um sistema de inspeção baseado em OPC UA onde o servidor possui em seu espaço de endereçamento as informações sobre os objetos inspecionáveis, localização, questões, formulários e usuários.

Na primeira versão do sistema existia um cliente OPC UA baseado em ASP .NET para realizar o gerenciamento do servidor sendo possível monitorar e adicionar informações no espaço de endereçamento do servidor, logo após foi desenvolvido um cliente OPC para a plataforma android para que este fosse o cliente instalado nos dispositivos móveis dos fiscais, porém este seria responsável apenas pela realização da inspeção.

O LIEC-Inspections é um sistema de gerenciamento de inspeções. Nele é possível a criação de novos objetos inspecionáveis, que representam algum equipamento ou instalação do laboratório. Para realizar a inspeção são criadas questões que direcionam o fiscal para observar determinados pontos de interesse do objeto. É possível agrupar as questões em formulários, onde estes são vinculados aos objetos, então os objetos podem possuir diversos formulários dependendo da finalidade da inspeção.

O estudo do sistema iniciou-se com o entendimento da modelagem de dados utilizada no desenvolvimento do sistema, qual o significado de cada classe e como elas se relacionavam. Temos a classe que modela o usuário, possui todas informações pessoais do funcionário e estes são classificados em 3 tipos: administradores, supervisores e fiscais.

Os demais modelos são referentes à inspeção em si, podemos observar na figura 12 como estas se relacionam. O objeto de inspeção possui a informação sobre sua localização e quais formulários são vinculados a ele, podendo ser vinculado a diversos formulários, já os formulários são constituídos por questões. A inspeção de um objeto consiste na resposta das questões contidas no formulário escolhido.

Figura 12 – Estrutura de relacionamento modelos referentes a inspeção do LIEC-Inspections.

Fonte: autoria própria.

O servidor do sistema LIEC-Inspections e o cliente baseado em ASP .NET foram desenvolvidos com o SDK da Unified Automation em sua versão para .NET Framework, sendo dependentes da plataforma Windows para serem utilizados. Visto isso foi pensado no desenvolvimento de uma interface que conseguisse comunicar-se com o servidor já existente porém que fosse independente da plataforma, tornando possível a criação de aplicações Windows, Android e iOS com a mesma interface.

Foi necessário então listar todas as funcionalidades que já existiam no cliente ASP .NET que deveriam fazer parte da nova interface, após o levantamento foi discutido com os professores do laboratório sobre a nova interface a ser criada e a proposta foi aceita.

4.2 SDK da OPC Foundation

Com a interface aprovada, buscou-se um SDK baseado em .NET Standard para que a nova interface fosse compatível com diversas plataformas, das opções disponíveis o SDK da OPC Foundation foi selecionado por estar disponível de forma gratuita em seu repositório público e pela sua garantia de suporte, pois a OPC Foundation é a organização que mantém o padrão.

No mesmo repositório do SDK há diversos exemplos de servidores e clientes utilizando o próprio SDK, o que ajudou no desenvolvimento da nova interface, foi utilizado o exemplo do cliente implementado em Xamarin Forms. Este exemplo trata-se de aplicativo simples para navegação no espaço de endereçamento, leitura e escrita de nós e monitoramento de nós.

4.3 Implementação da interface

É padrão do laboratório o desenvolvimento de soluções utilizando o Windows Forms, por este motivo foi necessário dedicar parte do estágio para aprender a utilizar esta ferramenta. Como é uma ferramenta utilizada há bastante tempo no laboratório os professores possuíam diversos materiais para auxiliar em sua aprendizagem e ela se deu em paralelo ao desenvolvimento da interface.

A interface foi desenvolvida em um projeto do tipo biblioteca de classes, para que este fosse utilizado pelas demais aplicações como uma *dynamic-link library* (DLL). Ela foi dividida em 4 classes, são estas:

- ***ClientConfiguration***: Métodos de configuração do cliente OPC;
- ***ClientConnection***: Métodos referentes à conexão;
- ***ClientDataAccess***: Métodos de acesso aos dados;
- ***ClientMethod***: Métodos que chamam os métodos do servidor OPC UA.

Iniciou-se o desenvolvimento pelas classes de configuração e conexão com o servidor, em sua primeira versão as configurações do cliente eram fixadas via código e o endereço do servidor ao qual a conexão seria realizada não poderia ser alterado durante a execução.

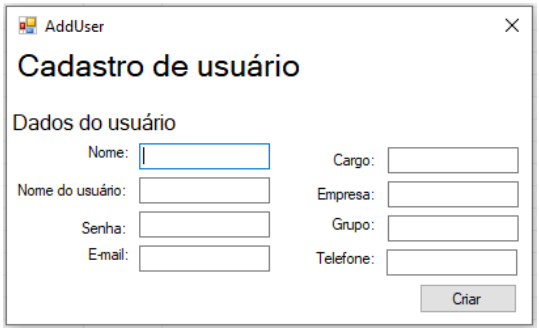
Quando foi possível estabelecer uma conexão com o servidor, começou a implementação da classe *ClientMethod*, visto que o servidor de testes haviam apenas 2 objetos de inspeções cadastrados e a adição de objetos ao servidor é realizada por meio dos métodos disponíveis neste. O primeiro método implementado foi o referente à criação de usuários e sua respectiva tela em Windows Forms para validação como podemos observar na figura 13a.

Como foi visto na figura 12, não é possível criar um formulário caso não haja questões no servidor, por este motivo o próximo método a ser implementado foi para adição de questões, na figura 13b é possível ver a tela criada para realizar a operação.

Para a criação de novos formulários é necessário ter informações das questões que foram cadastradas no servidor, foi necessário começar a implementação da classe

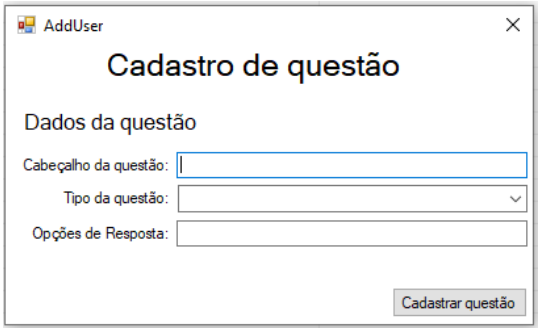
Figura 13 – Adição de usuário e questões.

(a) Tela para adição de usuários.



Fonte: autoria própria.

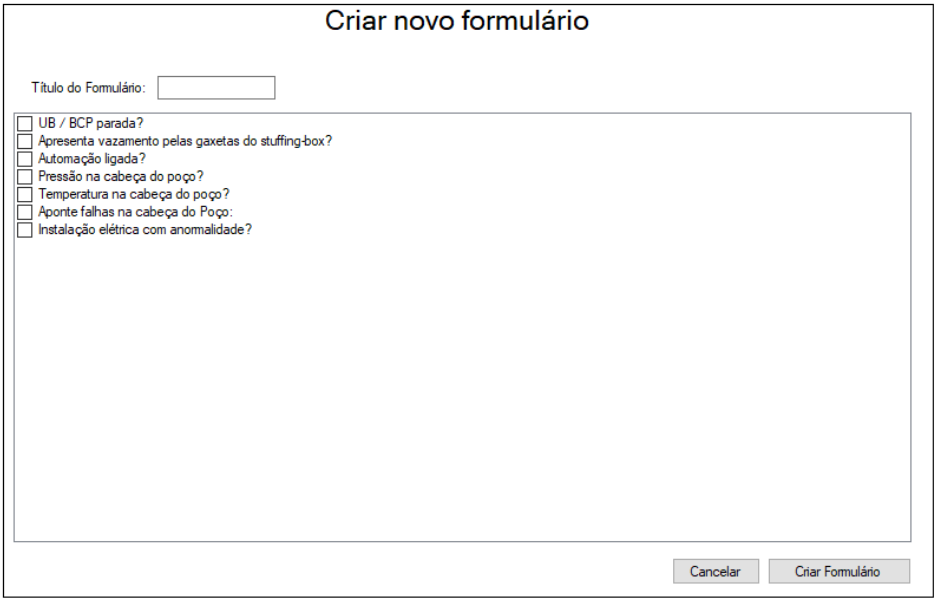
(b) Tela para adição de questões.



Fonte: autoria própria.

ClientDataAccess. A figura 14 mostra a tela de cadastramento de formulários observa-se a presença das questões do servidor e um campo para nomear o formulário criado .

Figura 14 – Tela para criação de formulários.

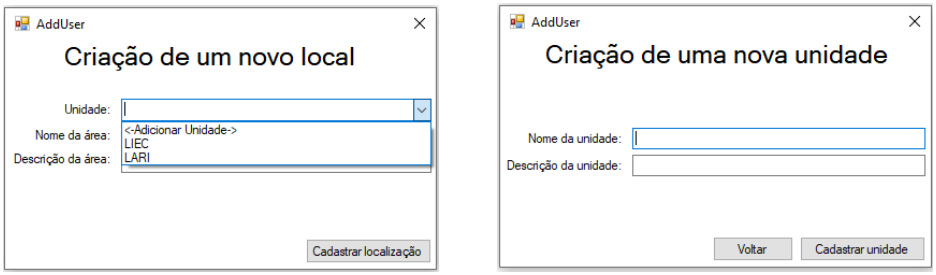


Fonte: autoria própria.

Há dois tipos de locais, unidade e área, onde as áreas fazem parte de uma unidade. Na figura 15a pode-se observar que para a criação de uma área é necessário associá-la a uma unidade, caso a unidade desejada não esteja cadastrada, podemos criá-la na tela mostrada na figura 15b.

Figura 15 – Adição de localização.

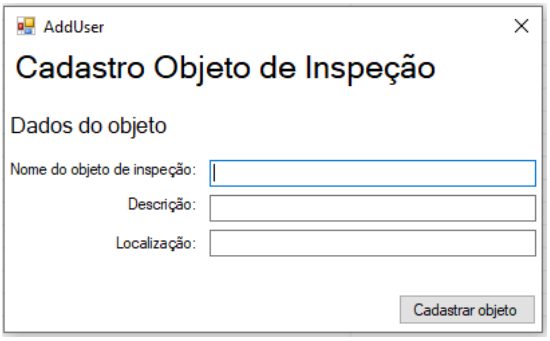
(a) Adicionar nova área. (b) Adicionar nova unidade.



Fonte: autoria própria.

Para a criação de um novo objeto inspecionável é necessário apenas as informações sobre seu nome, sua descrição e onde o objeto está instalado, como pode ser visto na figura 16.

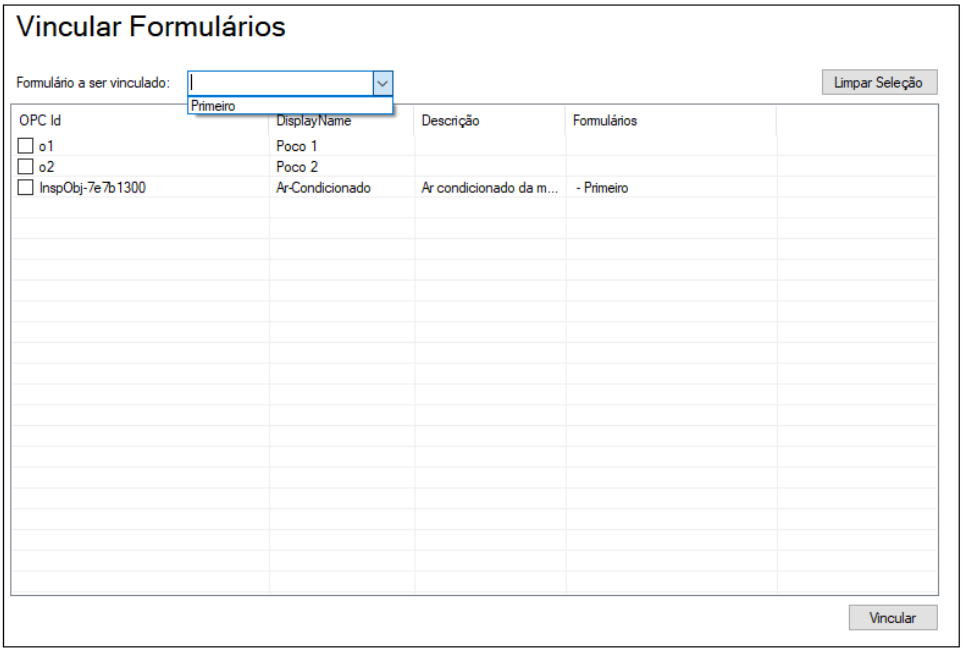
Figura 16 – Tela para criação de objetos inspecionáveis.



Fonte: autoria própria.

Restando apenas o métodos de vinculação de formulários a objetos para finalização da classe *ClientMethod*, para a implementação desde método foi necessário voltar para a classe *ClientDataAccess*, pois a vinculação de formulários necessita da informação dos formulários e objetos de inspeção cadastrados. Na figura 17 pode ser observado que é possível vincular um formulário a diversos objetos em uma mesma operação.

Figura 17 – Tela para vinculação dos formulários aos objetos.



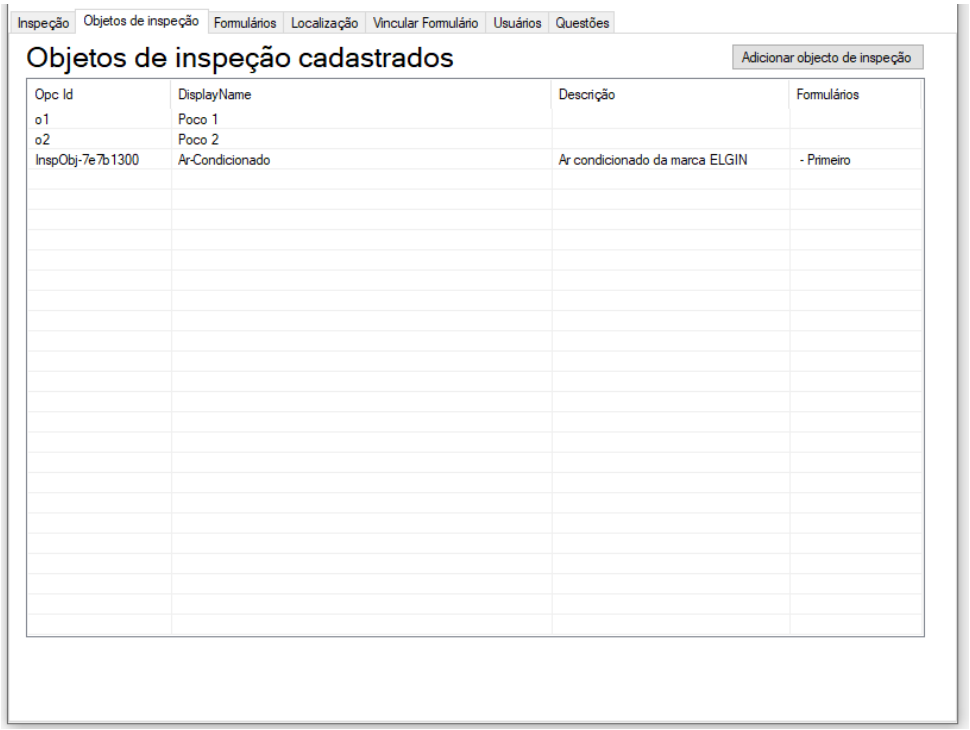
Fonte: autoria própria.

Com a classe *ClientMethod* finalizada restava apenas a implementação dos métodos de acesso a dados que não eram utilizados nas telas anteriores. Neste momento a interface do programa foi estruturada com o auxílio de abas na parte superior do mesmo, como pode ser visto na figura 18 onde podemos ver os objetos cadastrados e o botão para mostrar a tela para adição de novos objetos.

Em seguida, na figura 19 temos a tela para visualização dos formulários.

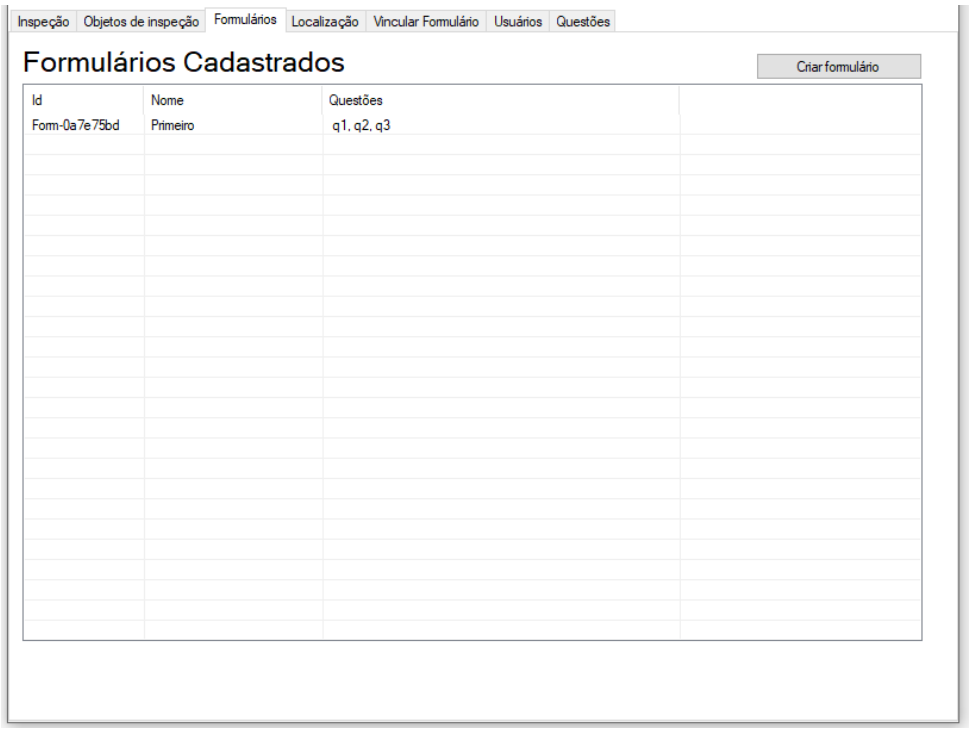
Para a visualização da localização utilizou-se uma estrutura de hierarquia, uma vez que unidades contém áreas, como podemos ver na figura 20.

Figura 18 – Visualização dos objetos inspecionáveis.



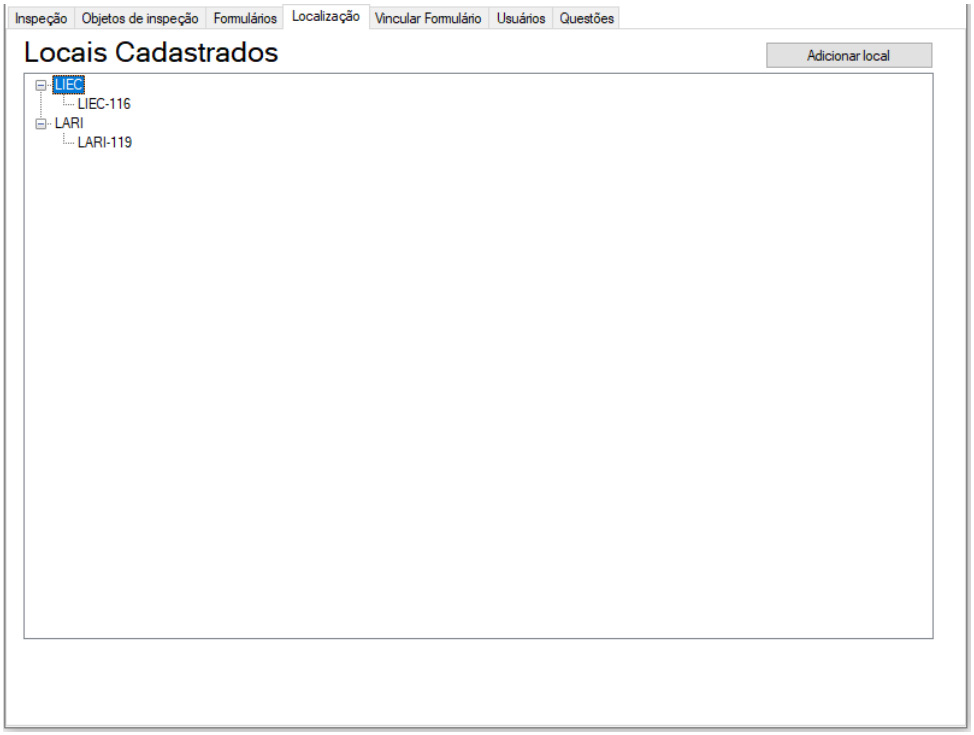
Fonte: autoria própria.

Figura 19 – Visualização dos formulários existentes.



Fonte: autoria própria.

Figura 20 – Visualização dos locais cadastrados.



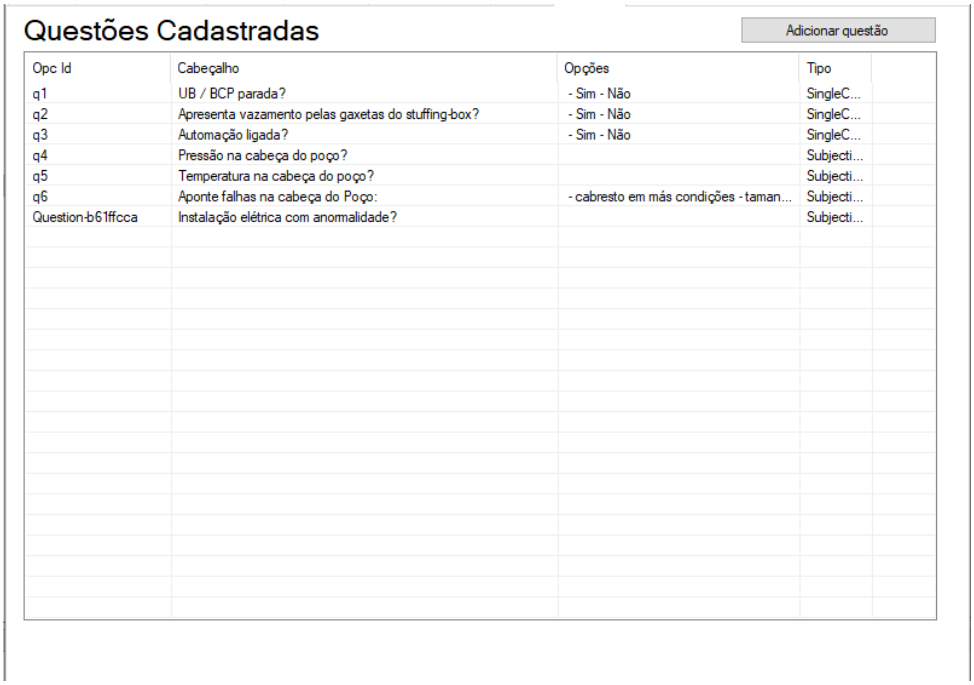
Fonte: autoria própria.

E por fim temos a tela criada para a visualização das questões existentes no servidor mostrada na figura 21.

Finalizados os métodos de acesso a dados e para utilização do métodos do servidor o último método implementado foi para a realização da inspeção em si, uma vez que este necessitava da visualização dos objetos inspecionáveis, formulários referentes ao objeto selecionado e por fim mostrar ao usuário as questões do formulário escolhido, como pode ser visto na figura 22 onde as respostas das perguntas devem ser separadas por ponto e vírgula.

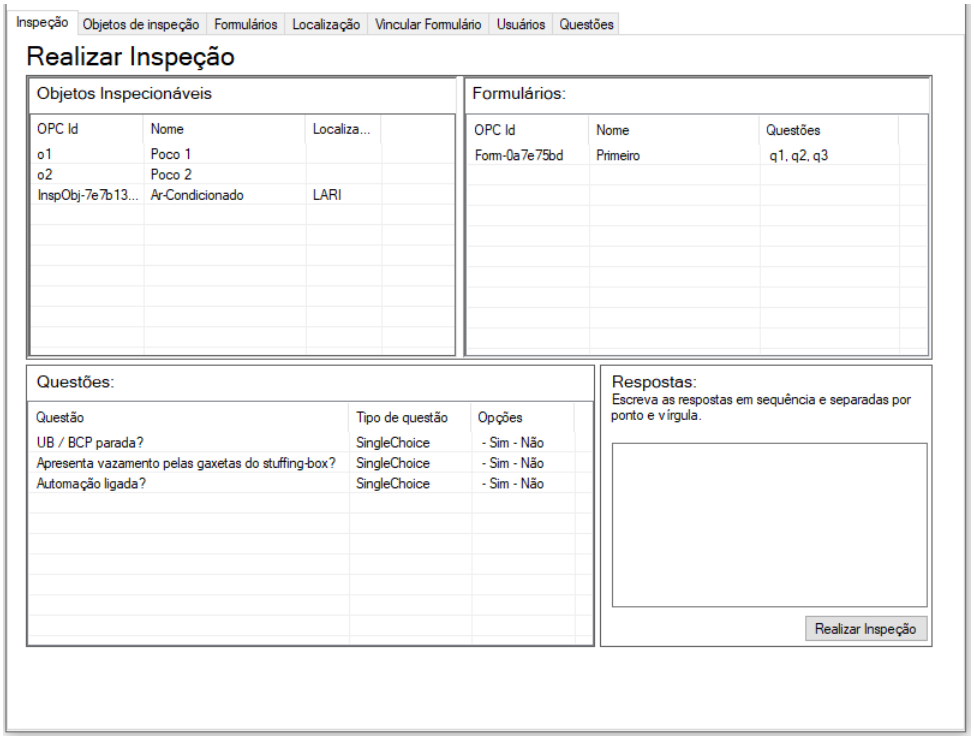
Após a implementação de toda interface foi solicitado que as configurações do cliente OPC, endereço do servidor fossem configuráveis pelo próprio programa e que houvesse um mecanismo para aceitação de certificados. Para a adição destas novas especificações utilizou-se um menu na parte superior da tela onde temos a opção de acessar as configurações da aplicação e caso estejamos conectados ao servidor, é possível se desconectar. Na figura 23 temos a tela de login com o menu superior, onde este é acessível em toda a execução do programa.

Figura 21 – Tela para exibição das questões.



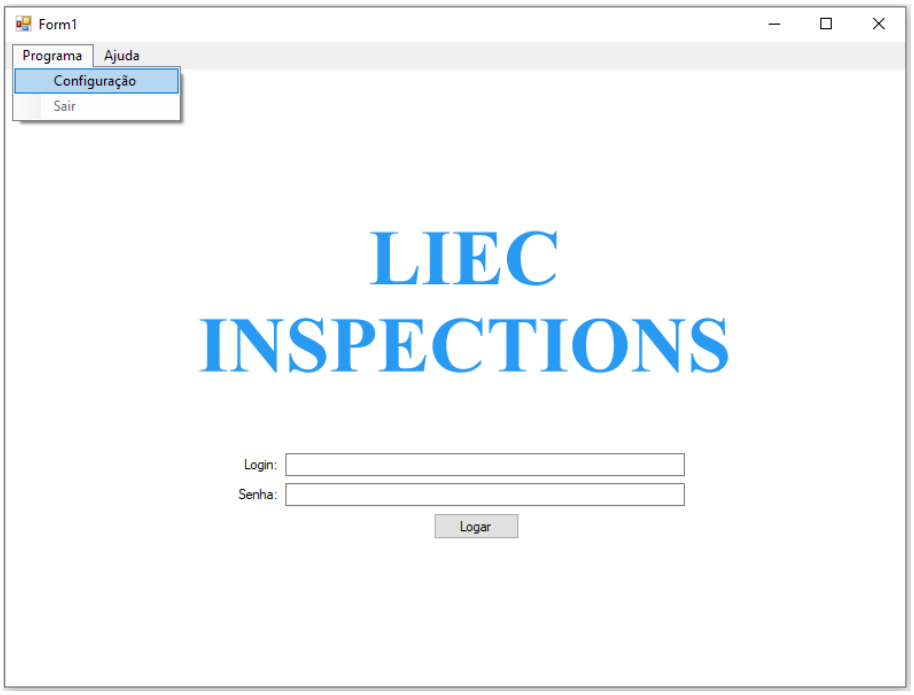
Fonte: autoria própria.

Figura 22 – Tela para realização de inspeções.



Fonte: autoria própria.

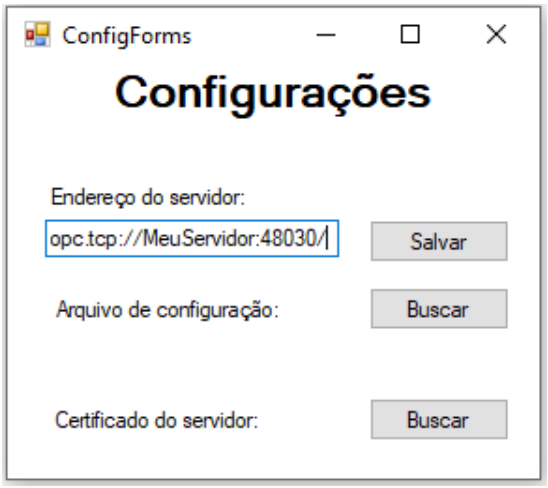
Figura 23 – Tela para realização do login.



Fonte: autoria própria.

Ao clicar na opção de configuração do menu o usuário é direcionado para a tela vista na figura 24, onde é possível alterar o endereço servidor, buscar o arquivo xml de configuração do cliente e selecionar o certificado para adicioná-lo na lista de certificados confiáveis.

Figura 24 – Tela de configurações do cliente.



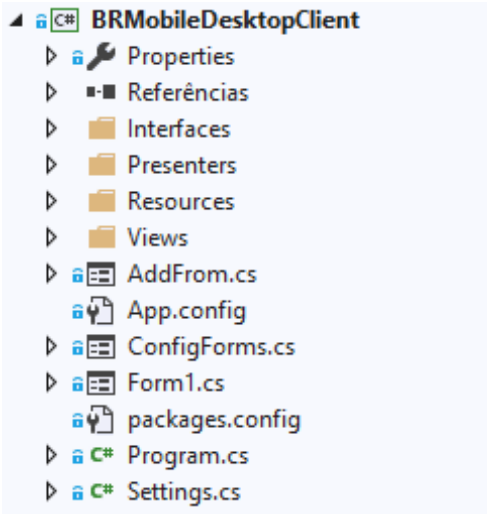
Fonte: autoria própria.

Após a finalização da interface começou a estruturação do programa, seguindo

o padrão de arquitetura de software MVP. A estruturação foi realizada posteriormente, pois o foco do trabalho foi o desenvolvimento da interface OPC UA utilizando um SDK para .NET standard, utilizar o MVP desde o início do programa aumentaria o tempo de criação de uma tela para validação da interface, o que atrapalharia a criação da mesma.

Durante todo o desenvolvimento da aplicação Windows Forms as *views* (telas) foram criadas com a utilização de *UserControl*, o que facilitou o procedimento. A estruturação teve seu início pela tela de login, posteriormente foi realizado pelos modelos onde a adaptação ocorreu primeiramente na tela de visualização seguida da tela de criação. Na figura 25 é possível verificar a estrutura do código.

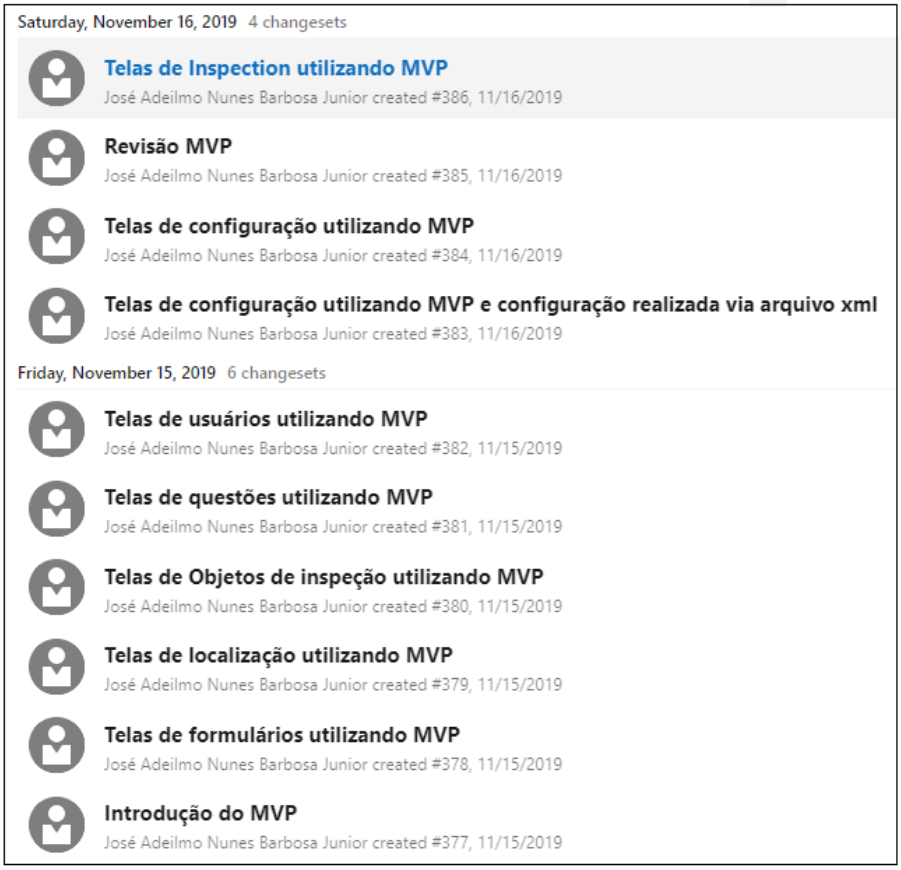
Figura 25 – Estrutura do código após adequação do código para MVP.



Fonte: autoria própria.

É possível observar na figura 26 a evolução da estruturação do código no controle de versão utilizado.

Figura 26 – Evolução da adaptação do código vista do controle de versão.



Fonte: autoria própria.

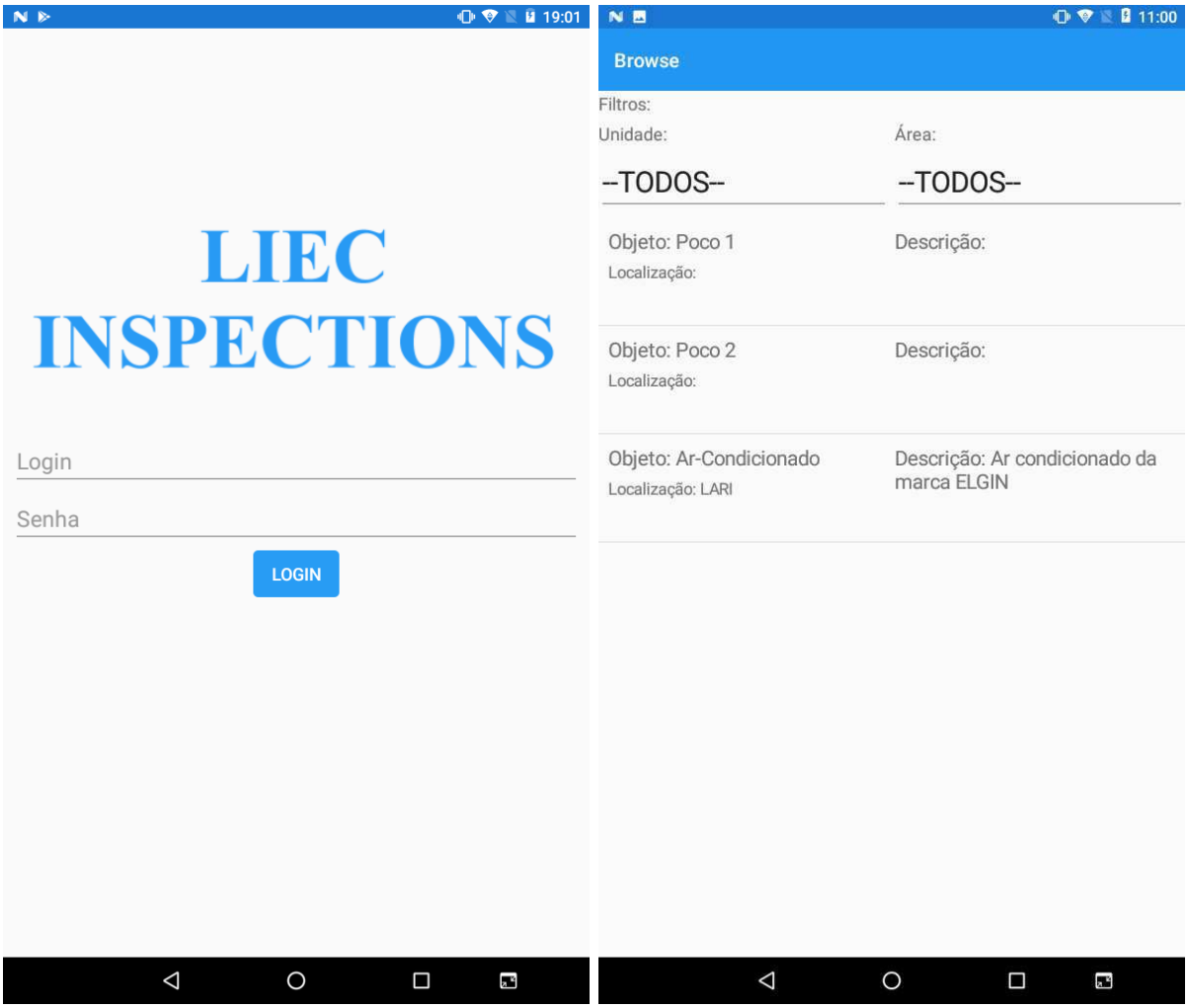
4.4 Xamarin Forms

Após a finalização da aplicação Windows Forms, foi solicitado a criação de uma aplicação em Xamarin Forms para a realização de inspeções, onde utilizaria um subconjunto dos métodos existentes na interface desenvolvida.

O Xamarin Forms utiliza como padrão de arquitetura de software o MVVM, sendo necessário realizar pequenas alterações ao reaproveitar o código do Windows Forms, já que o mesmo estava em MVP. O maior esforço foi na elaboração das telas, foi criado um protótipo da aplicação mobile, porém não foram implementadas todas, apenas foram implementadas telas simples referentes à inspeção que são mostradas nas figuras 27a, 27b, 28a e28b.

Figura 27 – Telas da aplicação Xamarin.

(a) Realizar login. (b) Objetos inspecionáveis existentes.

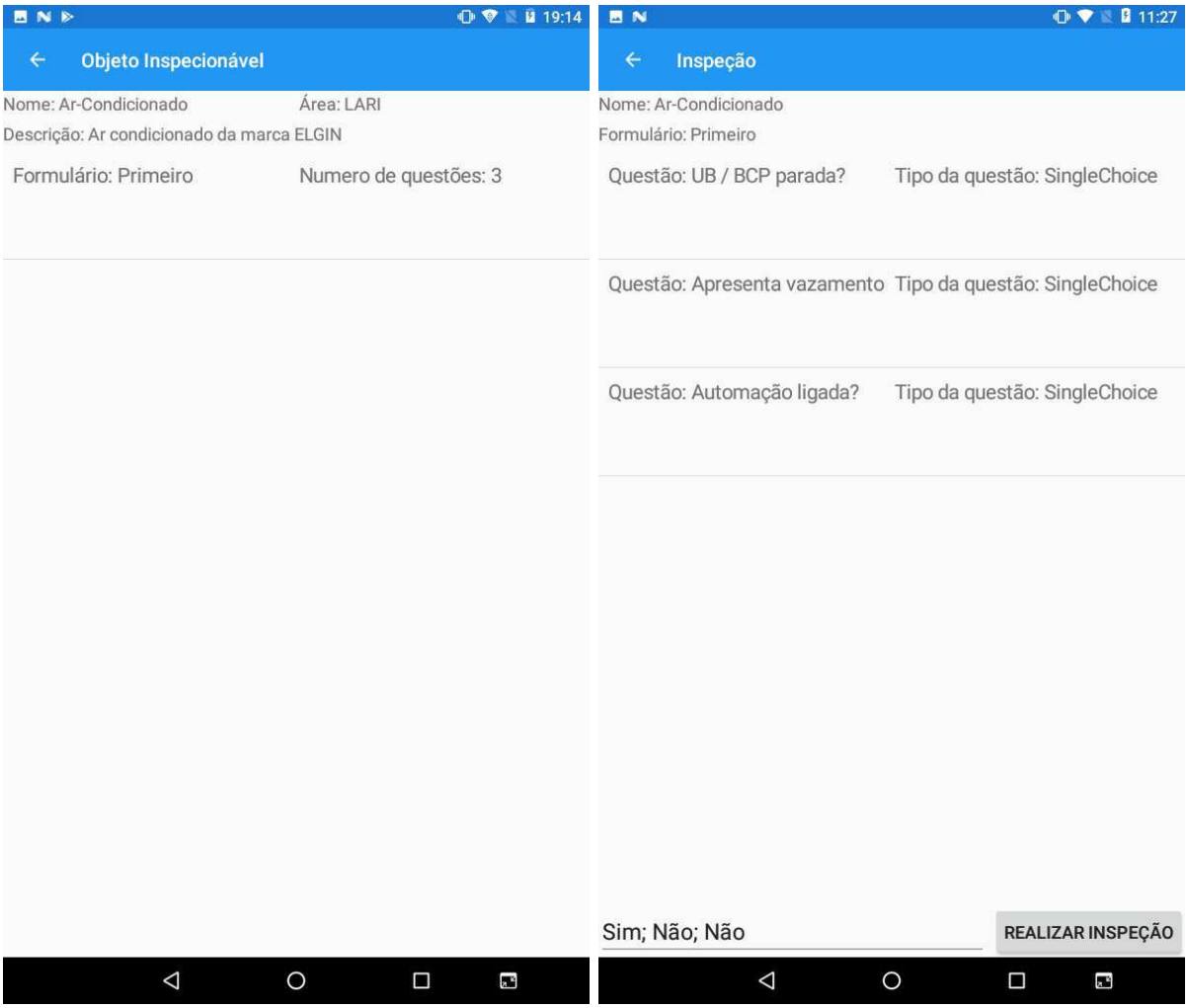


Fonte: autoria própria.

Figura 28 – Telas da aplicação Xamarin.

(a) Detalhes do objeto inspecionável e formulários vinculados.

(b) Realização de inspeções.



Fonte: autoria própria.

5 Considerações Finais

Durante a execução do estágio supervisionado percebeu-se sua extrema importancia, oferecendo ao aluno a possibilidade de por em prática conteúdos de disciplinas da sua graduação e aprofundar o conhecimentos em assuntos apresentados superficialmente. Aumentando a autoconfiança dos estudantes de graduação, pois as experiências em sala de aula fazem o aluno ter a sensação de que não se está aprendendo o que é necessário para trabalhar.

Neste período foi possível aplicar conhecimentos de disciplinas como: Sistemas de Automação Industrial, no conhecimento do padrão OPC UA e suas aplicações; Informática Industrial, no que diz respeito à engenharia de software; Técnicas de Programação, com o conhecimento adquirido para programação orientada a objeto.

Além da confiança adquirida pelo aluno, os conhecimentos aprofundados em desenvolvimento de software, padrão OPC UA e manutenção preditiva o fizeram mais preparado para o novo mercado de trabalho advindo da indústria 4.0. A experiência em desenvolvimento de aplicações é desejável, manutenção preditiva está mais presente e o OPC UA já é largamente utilizado na indústria em sua rede para conectar o maquinário já existente e graças a sua especificação 14 onde é definido um modo de operação *publisher-subscriber* é uma alternativa para aplicações em IoT industrial.

Por fim, o trabalho abre portas para o desenvolvimento de outras arquiteturas sendo possível adicionar conceitos de computação em nuvem e computação em borda para utilizar os dados gerados pelas inspeções para um melhor gerenciamento e aprimoramento das aplicações desenvolvidas.

Referências

COLLINS-SUSSMAN, B.; FITZPATRICK, B.; PILATO, C. *Version Control with Subversion*. O'Reilly Media, 2004. (O'Reilly Series). ISBN 9780596004484. Disponível em: <https://books.google.com.br/books?id=v1rN2MJ81JUC>. Citado na página 12.

IWANITZ, F.; LANGE, J. *OLE for Process Control: Fundamantals, Implementation and Application*. [S.l.]: Hüthig, 2001. Citado na página 6.

MAHNKE, W.; LEITNER, S.; DAMM, M. *OPC Unified Architecture*. Springer Berlin Heidelberg, 2009. (SpringerLink: Springer e-Books). ISBN 9783540688990. Disponível em: <https://books.google.com.br/books?id=de9uLdXKj1IC>. Citado 2 vezes nas páginas 9 e 10.

MCCONNELL, S. *Professional Software Development: Shorter Schedules, Higher Quality Products, More Successful Projects, Enhanced Careers*. Addison-Wesley, 2004. ISBN 9780321193674. Disponível em: <https://books.google.com.br/books?id=ylIAwQEACAAJ>. Citado na página 12.