



UNICAMP

Universidade Estadual de Campinas

Faculdade de Engenharia Mecânica

Desenvolvimento de um sistema Server/Client na plataforma LabVIEW para o laboratório FlowRs Lab

ES003 - Iniciação Científica II

Orientador:

Prof. Dr. Ricardo Augusto Mazza – FEM/Unicamp

Aluno:

Vinicius Allan da Silva

RA:

225295

Campinas - SP

14 de fevereiro de 2023

Sumário

1 Agradecimentos	2
2 Introdução	3
3 Revisão Bibliográfica	3
3.1 LabVIEW e suas vantagens	3
3.1.1 Estrutura do labVIEW	4
3.2 Protocolo TCP/IP e UDP: Vantagens e Desvantagens	4
3.2.1 TCP/IP dentro do LabVIEW	5
4 Desenvolvimento	8
4.1 Projeto	8
4.1.1 Server.vi, etapa check-status	9
4.1.2 Server.vi, etapa data	10
4.1.3 Listener_Paralelo.vi	12
4.1.4 Client.vi, etapa open-conection	12
4.1.5 Client.vi, etapa check-status	13
4.1.6 Client.vi, etapa data	14
4.1.7 My_IP.vi	15
4.1.8 Configuracao.vi	16
4.1.9 padrao_dados_ENTRADA.ctl	17
4.1.10 padrao_informacoes_meu_computador.ctl	18
4.1.11 Pega_Tags_cluster_entrada.vi	18
5 Resultados	19
5.1 Server	19
5.2 Client	20
5.3 Interface de Calibração	21
5.4 Testando a comunicação	22
6 Conclusão	24
7 Referências	25

1 Agradecimentos

Primeiramente agradeço a Deus pela saúde, e pelas oportunidades.

Agradeço ao meu orientador Prof. Dr. Ricardo Augusto Mazza pela paciência e disponibilidade e pelo convite para participação do projeto, também agradeço ao aluno de doutorado Matheus Pasquini que sempre esteve presente no laboratório me ajudando e me apoiando nos testes.

Agradeço aos meus pais pelo apoio e pela confiança e sou muito grato pela presença da minha namorada Barbara nos meus momentos difíceis que sempre tem me ajudado e me motivado a continuar.

2 Introdução

A proposta desse trabalho de Iniciação Científica é o desenvolvimento de um sistema Server/Client feito na plataforma LabVIEW que será a base para todas as comunicações de sistemas Supervisórios e demais aplicações, com os dispositivos Controladores Lógicos Programáveis no Laboratório Flow&Rs Labs.

Atualmente, dentro do laboratório, não existe uma centralização clara sobre a manipulação dos dados. Os computadores que desejam obter algum dado ou controlar alguma variável, precisa comunicar direto com os CLPs ou trocar dados entre as máquinas dentro de alguma aplicação labVIEW. O grande problema disso tudo é que existe limitações de acesso aos dispositivos CLPs além de gerar uma enorme bagunça sobre onde é gerado uma ação ou conversão sobre determinado dado.

Uma alternativa para isso é a centralização das comunicações através de apenas um computador, além do mesmo ser responsável pelas calibrações e configurações dos instrumentos, dessa forma, mudanças de configurações seria alterado em apenas um lugar e não seria necessário múltiplas conexões no mesmo dispositivo controlador. Com a ideia implementada, todas as outras aplicações que requerisse algum dado ou atuar sobre determinada variável, estabeleceria uma conexão com esse computador central e faria essa troca de dado.

O novo sistema a ser desenvolvido será dividido em duas partes principais:

O Server, que será o único a se comunicar com os CLPS, tanto ler como escrever em variáveis, além de ficar com a maior parte do processamento, como conversão de dados para valores de engenharia, seleção de variáveis a serem trocadas e monitoramento de programa Clients ligados tentando se comunicar com o servidor. O server só rodará em apenas um computador. E também o Client, que será uma parte do programa responsável pela conexão entre os dados e os programas supervisório presente no laboratório. O Client poderá ser rodado em múltiplas máquinas espalhadas pelo laboratório.

Toda a comunicação entre computadores será feita usando o protocolo TCP/IP.

3 Revisão Bibliográfica

Antes do sistema começar a ser desenvolvido, buscar as melhores ferramentas é uma etapa essencial para alcançarmos o nosso objetivo de uma forma mais eficiente possível, assim, entendermos o motivo da linguagem LabVIEW como a linguagem de desenvolvimento e um estudo sobre os protocolos TCP/IP e UDP é fundamental para sabermos qual direção tomar.

3.1 LabVIEW e suas vantagens

A palavra labVIEW é a junção da frase “Laboratory Virtual Instrumentation Engineering Workbench” ou seja, uma estação de trabalho virtual para instrumentação e engenharia.

O labVIEW acaba sendo um grande aliado para um processamento e análise dos dados de uma forma mais eficiente.

Diferente de outras linguagens de programação, o labVIEW tem uma linguagem G, ou seja, ela não é textual e sim gráfica, o que permite um rápido entendimento e compressão dos sistemas montados e um aprendizado mais intuitivo para os usuários. A plataforma conta com mais de 3 mil funções pré-montadas para ajudar o usuário. A plataforma é capaz de fazer integração com os hardwares do mundo físico juntamente com a aquisição dos dados, além de usar módulos matemáticos para uma análise estatística ou um pós-processamento dos dados.

3.1.1 Estrutura do labVIEW

Cada programa no labVIEW são chamados “VI”, isso porque eles se comportam como se fossem instrumentos virtuais.

Cada VI tem duas partes, o Painel Frontal e o Diagrama de Blocos. O painel frontal é a parte onde o usuário irá se comunicar com o programa, então as variáveis são mostradas de uma maneira fácil para que o usuário saiba o que fazer e como operar o programa. Já o Diagrama de Blocos é a parte que contém a lógica do sistema, nela há todo o fluxo de lógicas e acionamentos além de expressões matemáticas para o funcionamento do programa.

Os principais tipos de dados são os Strings, Integer, Float e Boolean. O LabVIEW é um forte aliado para trabalhar com Array e Cluster, que são estruturas de dados com múltiplos dados básicos dentro delas.

As sub-rotinas ajudam o código a ficar mais fácil, compacto e modular, e no labVIEW a utilização de sub-rotinas chama-se SubVI.

O labVIEW permite a utilização do código feito em uma VI dentro de uma outra VI o que facilita todo o processo.

3.2 Protocolo TCP/IP e UDP: Vantagens e Desvantagens

Para podermos decidir com qual protocolo iremos trabalhar, vamos entender como funciona os dois tipos de protocolos, suas vantagens e desvantagens.

De acordo com o site SpiceWorks.com TCP (Transmission Control Protocol) tem a característica de verificação dos dados enviados se eles foram recebidos, esse protocolo é muito utilizado dentro da internet global para troca de dados.

Diferentes dispositivos e programas na rede, se comunicam através desse protocolo, enviando pacotes de dados e assegurando que todos os dados foram entregues com êxito.

Todavia, antes de trocar o dado propriamente dito, o Server e o Client fazem uma conexão, o Server cria um canal de escuta e espera ativamente pela tentativa do Client de tentar se conectar, uma vez que essa conexão é feita, ela se mantém estabelecida e os dados podem ser trocados através dela, não necessitando de uma nova conexão. Por causa dessa característica de confiança, TCP é majoritariamente utilizado dentro da internet.

Entretanto, o TCP carrega alguns pontos negativos:

Por ser um protocolo confiável, ele necessita de 3 direções de confirmação, o que na prática pode significar alguma perda de tempo se compararmos a protocolos mais rápidos.

Em casos de perda de dados juntando com os sinais de verificação, podemos ter na prática um congestionamento de dados.

Agora falando um pouco sobre o UDP (User Datagram Protocol) que tem a característica de ser um protocolo rápido de transmissão de dados. Diferente do TCP, o UDP não cria uma conexão e a mantém, na verdade o Server fica encarregado de mandar os dados para o endereço a qual se encontra o Client, independente do dado ter sido comprometido ou não, e se a conexão existe, o Server sempre vai ficar enviando dados para o endereço de referência.

Como não há as etapas de verificação dos sinais, como no protocolo anterior, existe apenas uma direção dos dados e não há etapa de verificação, com isso o protocolo acaba sendo muito mais rápido e aliado para situações de comunicação de Tempo Real, ou comunicação com muitos pacotes de informações, porém, há chances de alguns dados serem perdidos no meio do caminho.

Elencando as principais diferenças entre os protocolos:

O TCP forma uma conexão de base, diferente do UDP que não forma uma conexão.

O TCP é mais propício a erros devido ao fato de ser mais exigente e confiável.

O TCP tem uma ordem de dados e respeita ela podendo atrasar as entregas de dados posterior, diferente do UDP que não se preocupa na ordem dos dados e acaba não sendo possível verificar se a ordem está correta.

O UDP acaba sendo mais rápido e mais eficiente. O TCP é indicado para conexões de ponto a ponto, para isso, caso com múltiplas conexões e transmissões é mais indicado o uso do protocolo UDP.

O TCP tem algoritmos para prevenir redes congestionadas e garantir a entrega dos dados, diferente do UDP que mantém enviando os dados pelo mesmo caminho.

O TCP é bem mais confiável que o UDP.

O TCP acaba sendo adequado para os casos em que a integridade dos dados são mais importantes do que a velocidade, Já o UDP, acaba sendo útil em transmissão de dados em tempo real e caso alguns dados sejam perdidos, a velocidade de transmissão se sobreponha.

Com isso, muitas aplicações online acabam utilizando TCP em conjunto com UDP para melhorar a eficiência sem perder muito da confiabilidade.

3.2.1 TCP/IP dentro do LabVIEW

O LabVIEW oferece funções prontas dentro da biblioteca TCP/IP de comunicação.

O LabVIEW também conta com exemplos simplificados e outros mais sofisticados sobre o assunto para melhor entendimento do desenvolvedor. O exemplo que foi consultado em um primeiro momento se encontra em : labview/examples/Data Communication/Protocols/TCP/Simple TCP/Simple TCP.lvproj.

A interface abaixo mostra um exemplo de troca de dados entre um Server e um Client que pode ser a base para o desenvolvimento do projeto, a troca de dado acontece praticamente em tempo real e com o gráfico mostrando o histórico dos dados aquisitados pelo Client, o sistema aparece ser robusto e não haver perdas de dados.

Veja a figura abaixo:¹

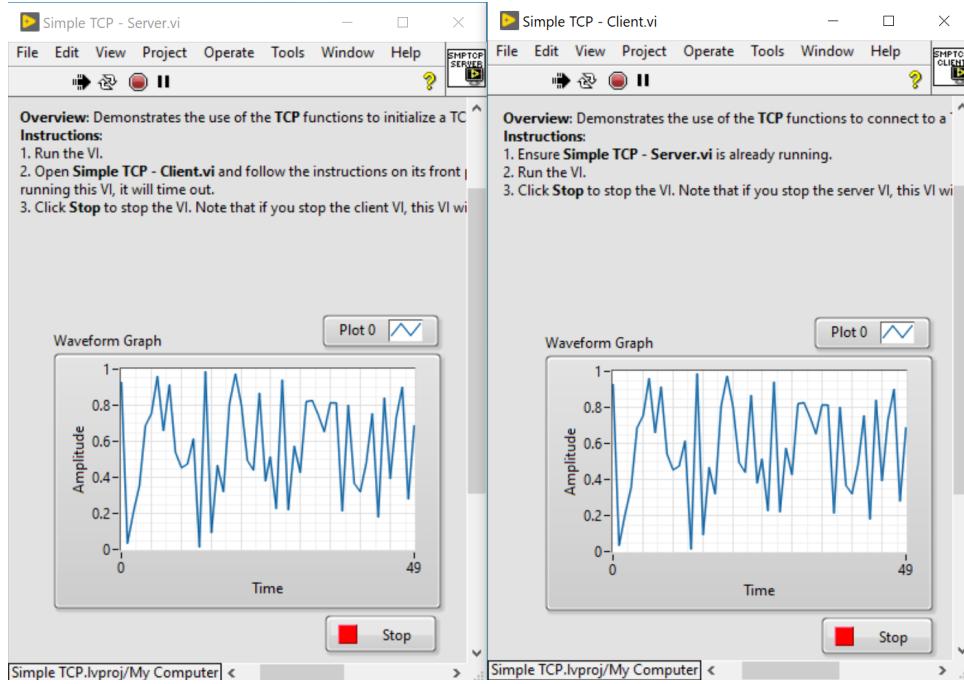


Figura 1: Interface Server Client exemplos do LabVIEW
Fonte:Própria

Todo Server necessita da função "Listener" que deve ser colocado no começo do programa, como mostrado na figura 2, ele cria uma porta local de escuta que no exemplo foi utilizado a 3335. Após isso, a função ficará a espera de uma tentativa de conexão por parte do Client. Apartir do momento que a conexão for feita com sucesso, o sistema entra no loop while, dentro desse loop deve ter as funções "READ" e "WRITE" para a escrita e leitura no Client. No exemplo 2 o Server está escrevendo dados gerados que foram gerados aleatoriamente.

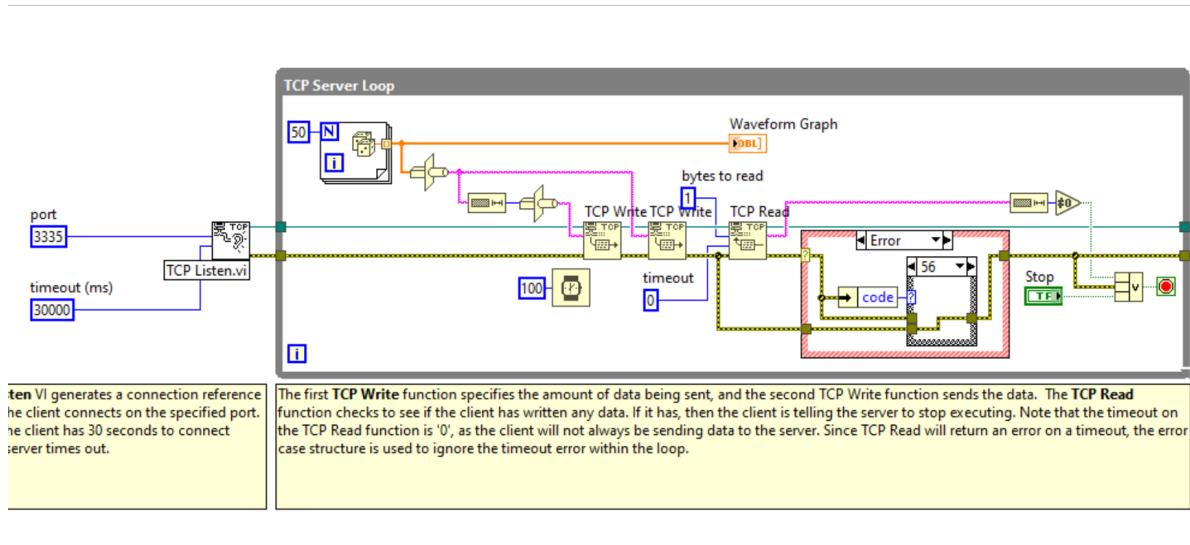


Figura 2: Diagramas de blocos do Server exemplo do LabVIEW

Fonte:Própria

Diferente do Server que não necessita saber quem vai se comunicar, o cliente precisa do endereço do server, pois ele irá tentar estabelecer através da porta virtual como pode ser visto na figura 3, o endereço é "localhost" pois as VIs estão sendo rodadas no mesmo computador.

Com a comunicação estabelecida, o sistema entrará no loop, e após isso, as funções "READ" são chamadas, primeiro é lido o tamanho da palavra e posteriormente, a palavra propriamente dita.

Esse sistema será a base para a comunicação Server/Client que será desenvolvida no laboratório.

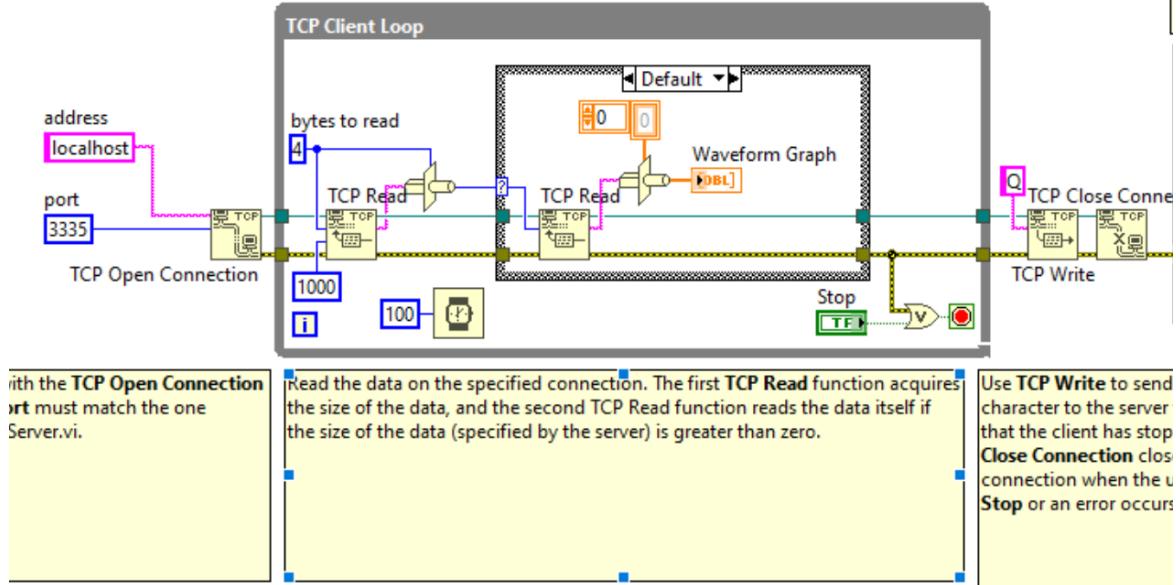


Figura 3: Diagrama de blocos do Client exemplo do LabVIEW

Fonte:Própria

4 Desenvolvimento

Apesar dos diversos arquivos bases e exemplos do LabVIEW para aplicações TCP/IP, o problema que enfretamos no laboratório requer um sistema próprio desenvolvido para as necessidades dos programadores e usuários locais.

A etapa de desenvolvimento é fundamental para ser feito o projeto no formato necessário para o laboratório além estroturar todo o sistema de uma forma mais eficiente e robusta possível.

4.1 Projeto

Para começar o projeto, foi necessário a criação de arquivo .lvproj que é o formato de projeto do LabVIEW, esse formato foi escolhido porque ao conectarmos nossas VIs no projeto, tudo fica mais organizado e dentro de uma hierarquia, o que facilita o entendimento da função de cada SubVI.

Como pode ser visto na figura abaixo 4, há diversas divisões dentro do projeto, como as VIs para Client, os padrões de variáveis, VIs relacionadas ao Server, e outras que não tem uma divisão clara.

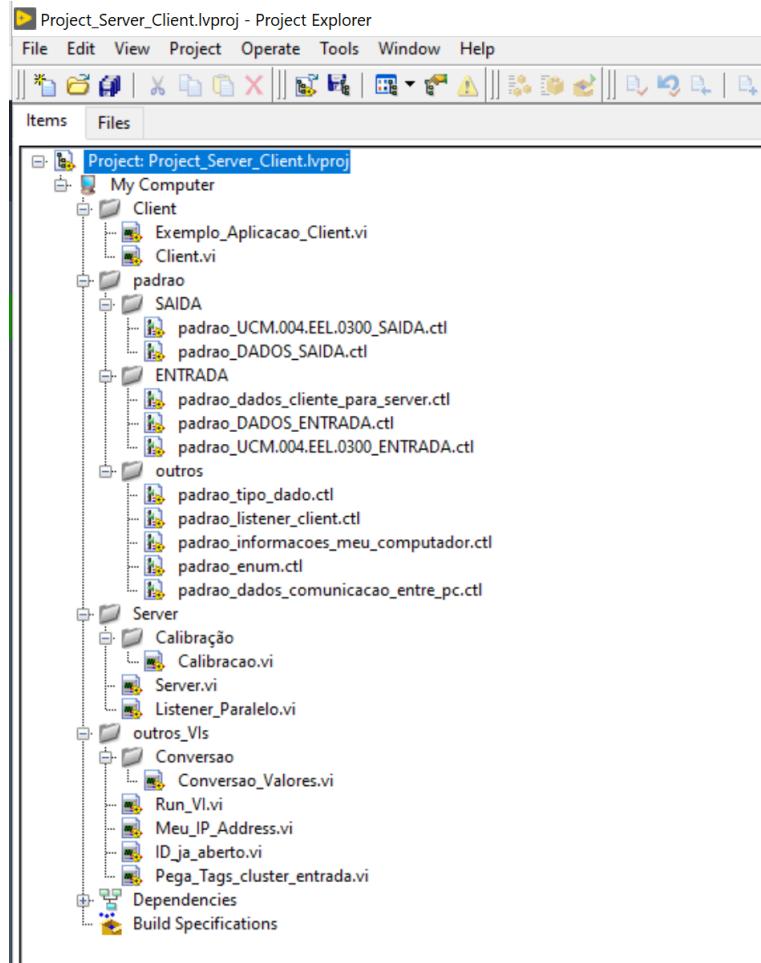


Figura 4: Projeto LabView, projeto_Server_Client

Fonte: Própria

4.1.1 Server.vi, etapa check-status

A primeira VI a ser criada foi a VI Server, a qual contém os principais mecanismo de comunicação entre o Client e reúne as principais funções para o funcionamento do sistema, ou seja, é a "Main" da parte do Server.

Dentro da VI Server, há duas princiais etapas importantes que foram desenvolvidas, a etapa de "check-status" e a etapa de "data".

Como pode ser visto a figura abaixo 5, a primeira coisa que a VI faz é rodar outra VI em paralelo chamada "Listener_Paralelo", o motivo dessa outra VI será explicado mais adiante. Após isso, o elemento "Ctrl Val. Get" recebe a referência da Vi que está rodando em paralelo e pega o valor da variável "Novo ID" que é um novo ID tentando se conectar ao computador Server, após isso, caso seja realmente um novo ID, e não repetição de um já aberto, ele é adicionado para o array de IDs conectados.

Nessa etapa "check-status" é também onde é atualizado o status de comunicação para o usuário, com isso, é feito uma análise sobre a quantidade de Clientes através do Array de IDs e caso ainda não haja nenhum Client conectado, é avisado ao usuário sobre a necessidade.

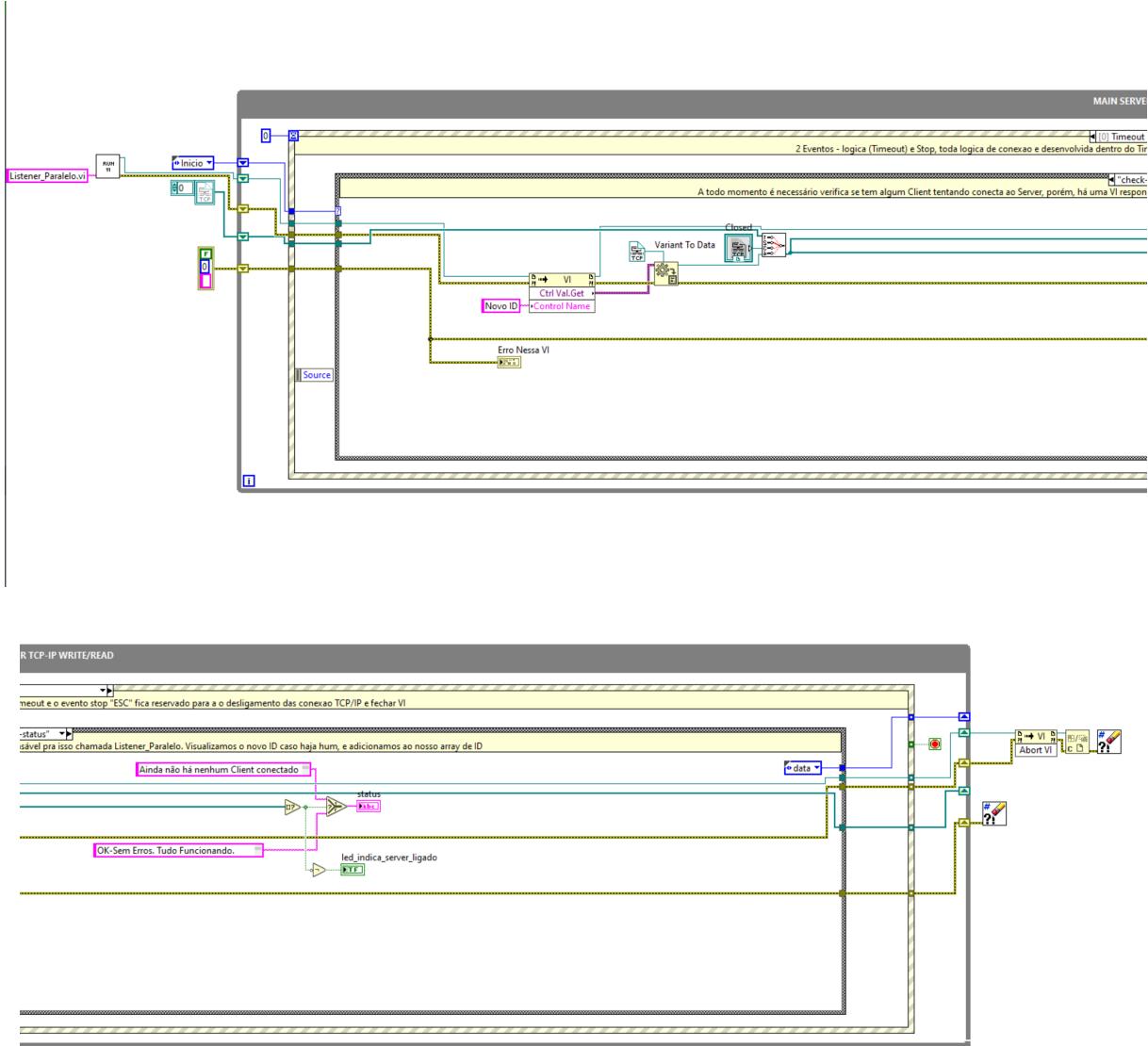


Figura 5: VI Server, etapa check-status

Fonte: Própria

4.1.2 Server.vi, etapa data

A outra etapa que o Server possui é a de "data", a qual constitui a alma do sistema, pois através dela os dados são enviados e recebidos do Client.

Nessa etapa, foi criado um loop do tamanho do array de Clients pois para cada Client vamos trocar informações diferentes. O cluster "todos_os_dados_SAIDA" contém um padrão de representação de todos os dados que serão saída do CLP, nessa etapa, esses dados foram passados para dentro do loop e logo em seguida, há uma etapa de conversão dos valores, pela SubVI "Conversao_Valores" baseado no tipo de dado que cada Cliente escolheu, ou seja, "Puro(mA) ou Engenharia". Após passar por essa conversão, o Cluster passa por uma função chamada "Flattern to JSON" ou seja, converte o formato de dado do LabVIEW para o

tipo JavaScript Object Notation, que é um formato compacto para troca de dados de forma simples e rápida entre sistemas que utiliza texto legível a humanos, no formato atributo-valor, isso tudo é necessário pois o tipo de comunicação que vamos utilizar é TCP/IP que só permite troca com String no LabVIEW, então essa etapa de conversão é fundamental. Com todos os dados já convertidos em String, utilizamos a função "TCP Write" para escrever tanto o tamanho da String a ser enviada, no primeiro momento, quanto para enviar os dados propriamente ditos. Na sequência, utilizamos as funções de "TCP read" para lermos os dados que estão vindo do Client, porém, também precisamos ler primeiro o tamanho da String a ser lido, e por final, a String de dados, como ela também foi convertida para JSON, precisamos revertê-la para nosso formato de Cluster, para isso utilizamos a função "Unflatten From JSON". Caso o sistema Client seja fechado no meio dessa etapa de troca de dados, a comunicação dará um erro, e é necessário fechar ID de comunicação, isso é feito dentro da janela do Case "Error". Se tudo ocorrer bem, os dados serão trocados com sucesso, e os Arrays "Clientes Conectados" e "IPs Conectados" serão atualizados com as informações reais dos computadores conectados. Veja a figura abaixo para melhor entendimento:

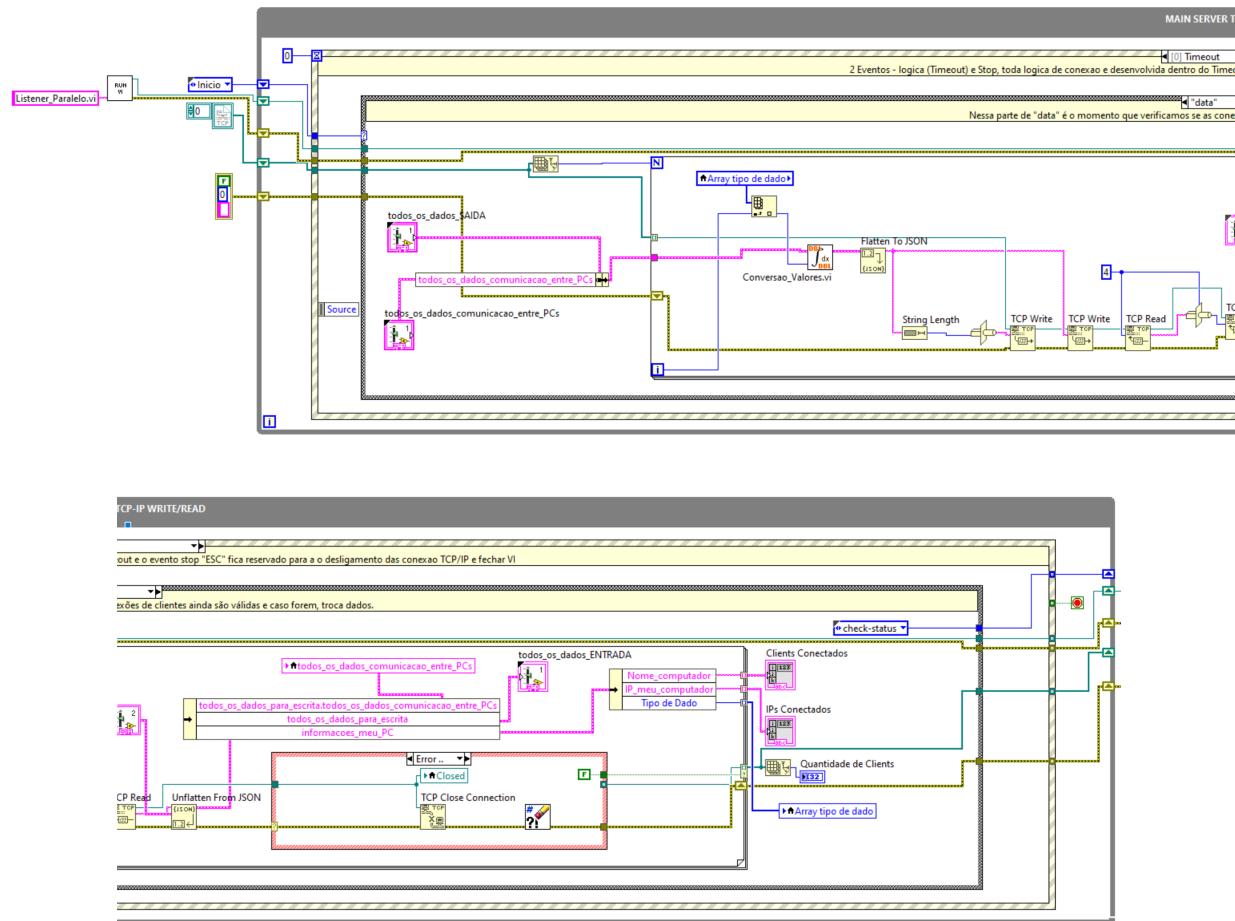


Figura 6: VI Server, etapa data

Fonte: Própria

Importante ressaltar que essas duas etapas, check-status e data, ficarão sempre rodando em loop e alternando entre elas.

4.1.3 Listener _ Paralelo.vi

Para comunicações TCP/IP, o Server, que será responsável pela comunicação de todos os computadores, ele precisa fazer a abertura do "Listener" ou seja, criar uma porta local para que todos que quiserem se comunicar com ele via TCP/IP precisará encontrar essa porta de uma maneira virtual. Também, é responsabilidade do Server ficar "ouvindo" se existe mais algum Client tentando se comunicar, porém, essa etapa consome um tempo razoável e como desejamos que a etapa de troca de dados seja feita de uma forma mais rápida e eficiente possível, essa VI foi criada de uma maneira a roda em paralelo com a VI Server principal. Como pode ser visto na figura abaixo, O "escutador" é criado na porta local 3335, e após isso, entra em um loop while onde utiliza a função "TCP Wait on listener" para tentar ouvir algum Client tentando se conectar ao computador Server. Essa etapa dura 100ms e após isso ele atualiza a variável "Novo ID" com um possível novo ID, caso não ocorra, será gerado um erro conhecido e o loop repetirá.

É através da variável "Novo ID" que a VI Server pega o novo ID tentando se conectar, como foi mostrado na seção Server.vi etapa "check-status".

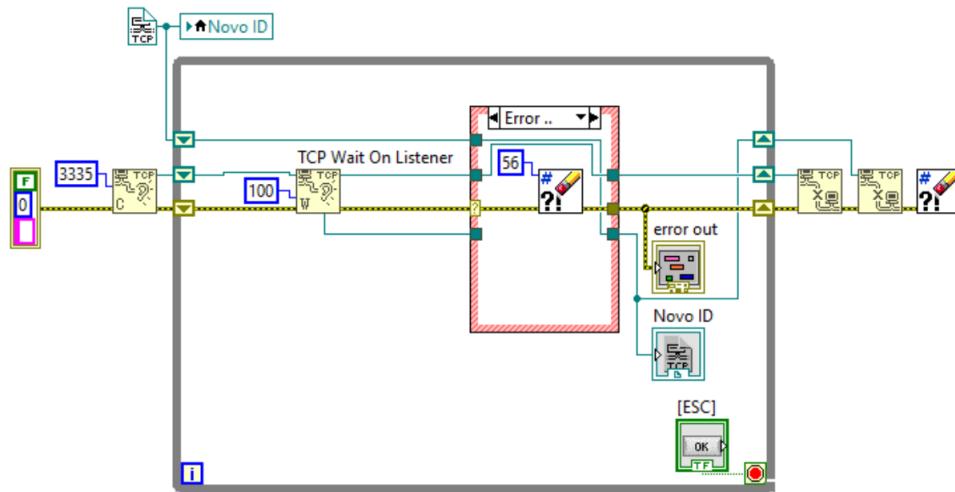


Figura 7: Listener _ Paralelo.vi

Fonte:Própria

4.1.4 Client.vi, etapa open-conection

O Client é a VI que irá rodar na máquina em que estiver rodando a aplicação para algum experimento, dessa forma, o Client tem uma responsabilidade muito grande no recebimento de dados, ou seja, visualizar as saídas do CLP, e no envio de dados, informações e valores para as entradas do CLP. Essa etapa é a responsável por fazer a conexão entre a máquina Server e a máquina Client. Como foi mencionado nas seções anteriores, o Server fica escutando para

ver se tem algum Client tentando se conectar e é nessa etapa que o Client faz o pedido da conexão. A função "TCP Open Connection" recebe o IP do Server, e a porta virtual a qual ela irá tentar se conectar, após 1 segundo tentando, caso não houver êxito, o usuário é avisado, como veremos mais a frente, noticiando que há um problema com o Server ou de rede.

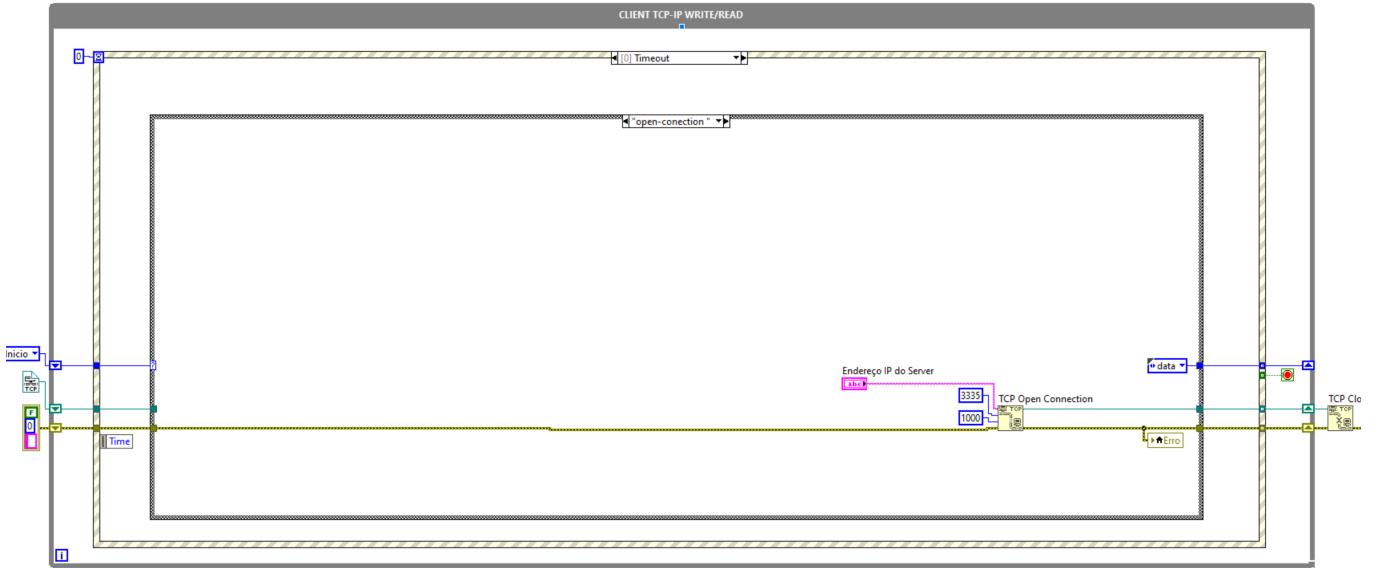


Figura 8: Client.vi, etapa open-conection
Fonte: Própria

4.1.5 Client.vi, etapa check-status

Nessa parte do programa, o objetivo é avisar o usuário o que está acontecendo e caso houver erros, instruí-los de como proceder, assim, toda vez que o programa passar por essa etapa, ele checa se há algum erro no Cluster de erro, caso houver e sendo um erro conhecido como por exemplo, Server não inicializado, o erro é limpo e uma mensagem é colocada na barra de status para o usuário saber qual providência tomar, por exemplo, o erro 56 é causado pela ausência do Server, assim, o status atualizado para o usuário é: "Problemas com o Server - Tempo limite para comunicação foi excedido. Verifique se o Server está ligado ou Tente Reiniciar o Server e o Client".

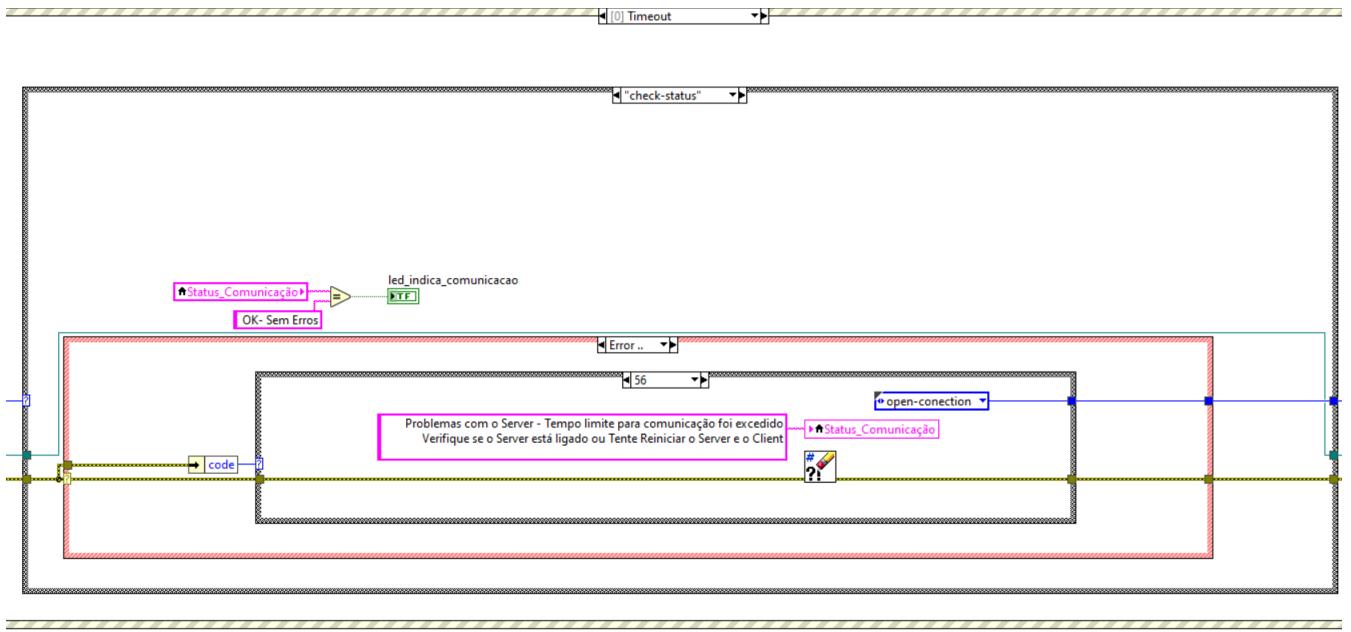


Figura 9: Client.vi, etapa check-status

Fonte: Própria

4.1.6 Client.vi, etapa data

A etapa "data" é a parte mais importante desse programa, visto que é o momento onde ocorre a troca de dados via TCP/IP. Como pode ser visto pela figura abaixo, o padrão é o mesmo da parte de troca de dados no Server. Os dados lidos em formato String pela função TCP Read são convertidos através da função Unflatten from JSON no formato do Cluster a qual era seu formato original. Logo em seguida, temos a parte de escrita de dados no Server a qual é o inverso do de leitura, os dados entrada presentes no cluster "todos_os_dados_ENTRADA" são adicionados ao cluster de nível acima chamado "dados_clientes_server" pois aproveitamos esse momento para adicionarmos informações do computador Client para o Server. Todas essas informações são convertidas em String no formato JSON e enviadas pela função "TCP Write".

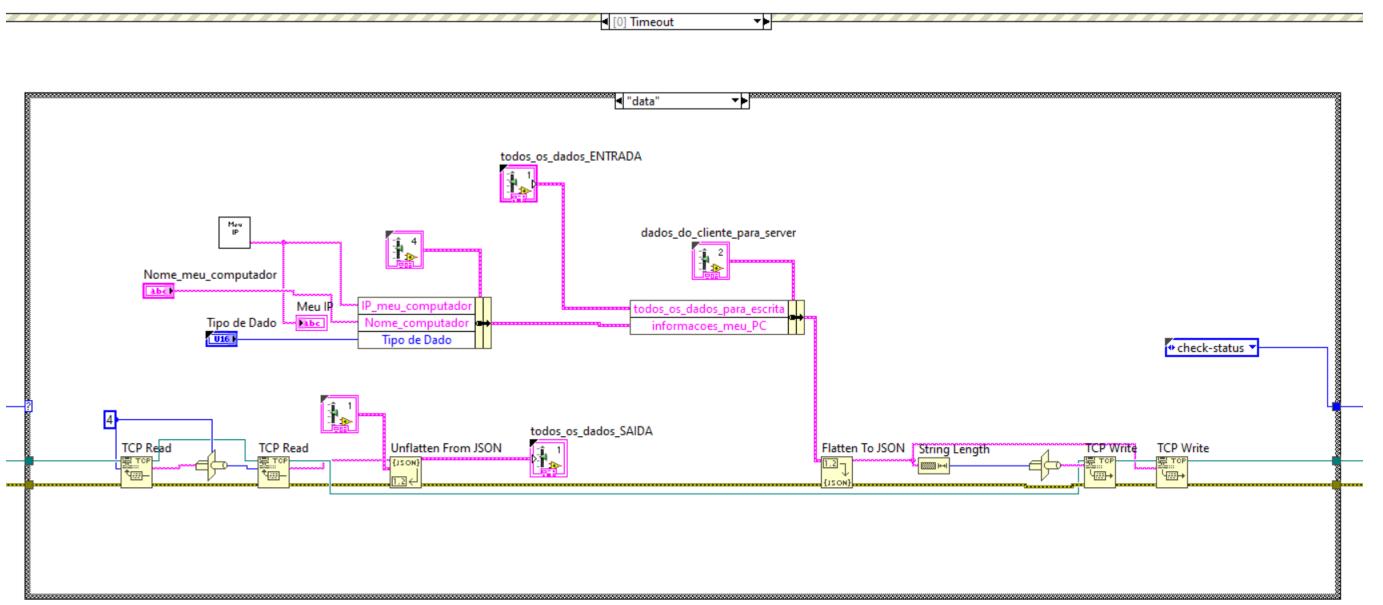


Figura 10: Client.vi, etapa data

Fonte: Própria

A etapa de "check-status" e "data" ficam sempre alternando entre si, porém, quando o erro 56 é detectado na etapa de "check-status", abre-se uma exceção e é chamada a etapa de "open-conection" para tentar uma nova conexão com o Server pois a mesma foi perdida.

4.1.7 My_IP.vi

Uma SSubVI muito importante é a VI My_IP.vi que pega o IP vigente da máquina a qual o programa está rodando, ela é utilizada para o Server saber quais os IPs que estão conectados a ele:

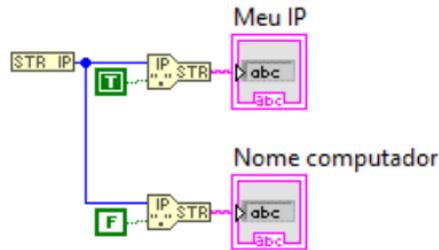


Figura 11: My_IP.vi

Fonte: Própria

4.1.8 Configuracao.vi

Com o objetivo de centralizar as informações e os cálculos, foi decidido que toda etapa de calibração dos instrumentos seria feito num local único, esse local é dentro do programa Server, dessa forma, caso fosse necessário mudar alguma calibração, todos os cálculos seriam feitos em um computador Server e corrigir ou encontrar o erro seria mais fácil. Foi criado então uma VI de calibração para centralizar toda essa etapa e melhorar o encapsulamento do programa.

Para facilitar esse processo e ter um banco de dados de configuração, o LabVIEW dispõe de um biblioteca para arquivos de configuração onde o mesmo é dividido sempre em seções e chaves "keys", as seções é o título do instrumento a qual será configurado, ou seja, a TAG do instrumento, e através dessa seção, podemos adicionar quantas chaves nós quisermos, por exemplo, podemos associar a cada seção, uma chave chamada "fabricante" ou "Mínimo" e "Máximo" e assim por diante, assim fica tudo mais organizado, pois quando quisermos obter uma informação sobre determinado Instrumento, basta buscarmos a TAG e junto virá todas as informações penduradas nas chaves respectivas a seção.

Como pode ser visto na imagem abaixo, o programa abre o arquivo .txt chamado "config_instrumento.txt" e pega todas as seções disponíveis, isso porque se trata da etapa inicial do programa onde é necessário visualizar qual seções já foram calibradas e quais foram adicionadas. Uma SubVI que lê todas as TAGs dos dados de entrada é chamada, vamos explicar essa VI mais pra frente, e é feito uma comparação sobre a existência daquela TAG no arquivo de configuração, caso essa TAG seja vinculada a um dispositivo do tipo Booleano, também não há interesse em adicionarmos ao arquivo de configuração. Caso seja encontrado um novo dispositivo, a nova seção é criada com as chaves em formato padrão para que o usuário possa adicionar no momento pertinente.

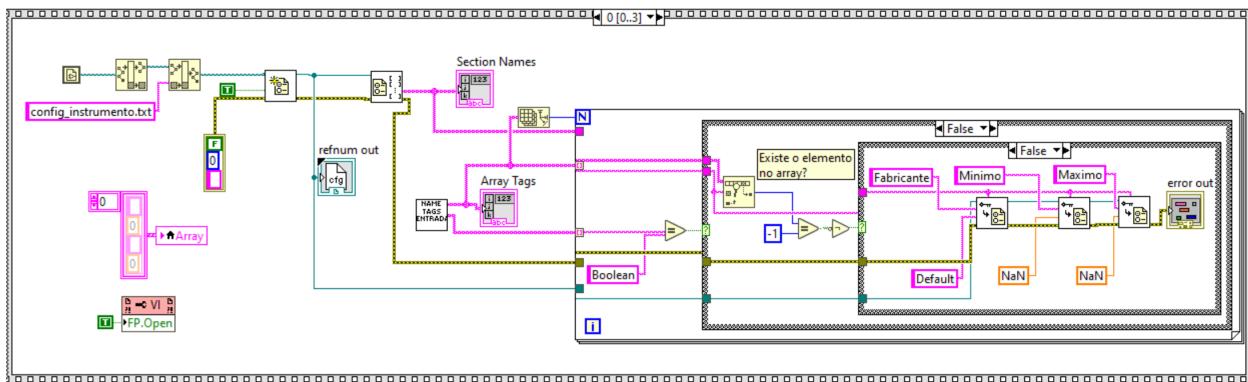


Figura 12: Configuração.vi

Fonte: Própria

4.1.9 padrao_dados_ENTRADA.ctl

No LabVIEW existe uma ferramenta muito importante para definição de padrões dentro do programa. Isso é necessário quando vamos utilizar um formato padrão de dado que irá se repetir e trocado entre múltiplas VIs, pois criando um padrão, você centraliza o formato e todas as utilizações desse tipo será uma cópia do padrão verdadeiro. No LabVIEW isso se chama "Type Definition".

A fim de melhorar a organização e o encapsulamento do programa, foram criados diversos padrões, principalmente para os dados de entrada e os de saída. No primeiro momento, criou-se um padrão Clusters de Entrada e um Cluster de saída, e dentro de cada desses clusters, padrões para o CLP da marca HiTecnologia P7C sendo eles: UCM.004.EEL.0300 e UCM.004.EEL.0200.

Os padrões foram separados em cluster de entrada e cluster de saídas, e eles serão vinculados as portas de I/O dos CLPs, através do computador Server com o OPC Servers. Na imagem abaixo, um tipo de padrão é mostrado e representa todos os dados de Entrada, como pode ser visto, a sub Cluster chamado "UCM.004.EEL.0300" com todas as variáveis de entrada desse dispositivo, vale ressaltar que, dentro desse cluster também há uma divisão entre sub clusters dividido entre "DIGITAL", "ANALOGICO"e "ModBus"para ajudar na separação no tipo de dado.

O cluster "todos_os_dados_comunicacao_entre_PCs"é um Cluster interno para troca de dados entre as máquinas que estarão rodando, eles não representam e nem serão vinculados com dispositivos físicos.

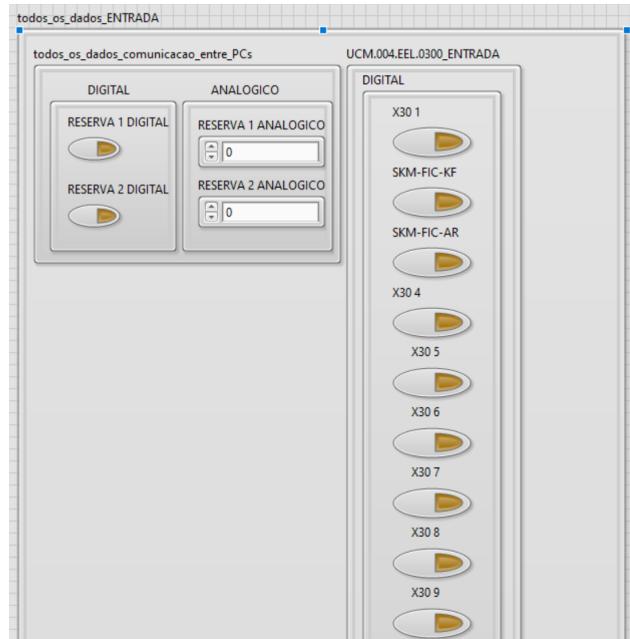


Figura 13: Type Definition, padrão para dados de entrada
Fonte: Própria

4.1.10 padrao_informacoes_meu_computador.vi

Outro padrão que foi necessário o desenvolvimento, foi o type definition do cluster com informações da máquina Client, esse padrão reúne as informações do Client que é importante o Server saber, o IP do computador é automaticamente gerado baseado na máquina Client através da SubVI "My_IP.vi". Uma informação muito importante é o tipo de dado, ou em outras palavras, qual a conversão, que o Client quer visualizar. A priori, há 2 tipo de conversão, "Engenharia"ou "Puro(mA)", assim, o Client poderá alterar a visualização para o tipo desejado.

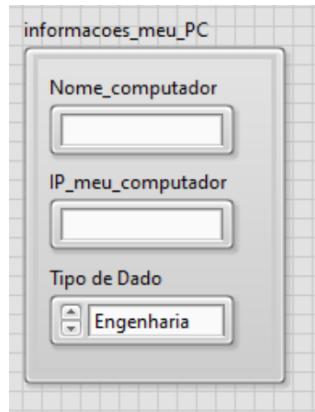


Figura 14: Type Definition, padrão das informações do Client
Fonte: Própria

4.1.11 Pega_Tags_cluster_entrada.vi

Outra VI que precisou ser desenvolvida, foi a VI que tem acesso aos tipos padrões de dados, ela acessa todas as TAGs e cria uma array com as TAG em sequência, essa VI é fundamental para o programa saber quais as TAGs que estão no programa e se alguma foi adicionada ou modificada.

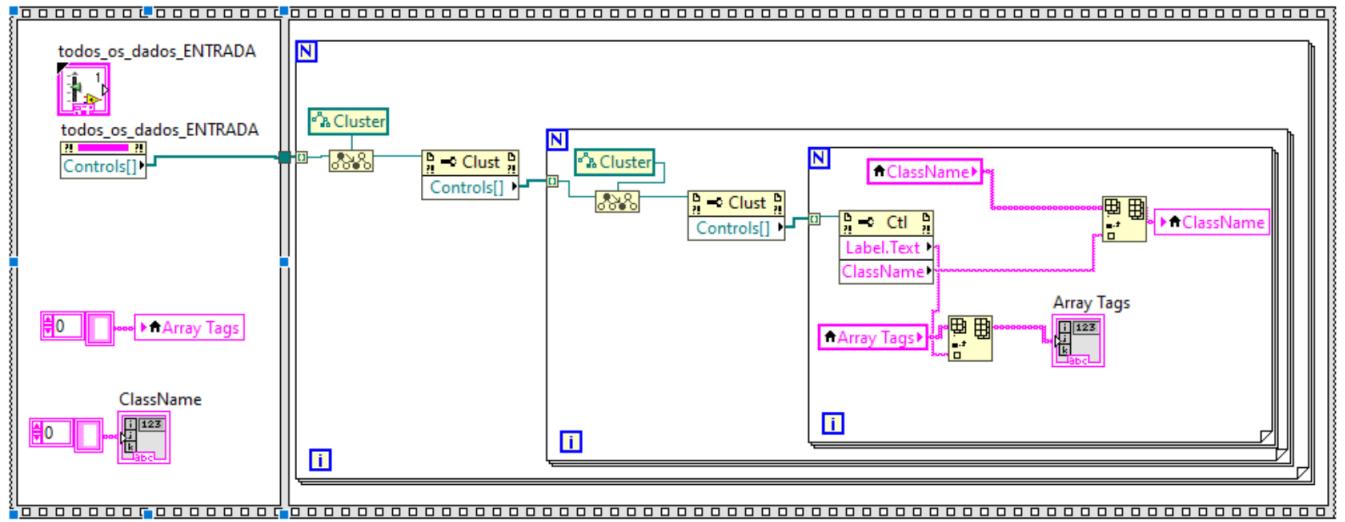


Figura 15: VI responsável por pegar as TAGs dentro dos clusters

Fonte: Própria

5 Resultados

Com o desenvolvimento finalizado, partiu-se para a parte de comissionamento para testar todo o código e detectar possíveis falhas.

5.1 Server

O objetivo da interface Server era ser a mais intuitiva e simples possível, para que o usuário que não seja desenvolvedor possa operá-la tranquilamente e em caso de erros, saiba como agir.

Foi definido a cor Laranja para o usuário acostumar com a coloração e nunca se confundir com outra VI, o LED Laranja aceso indica que o programa está rodando e que a comunicação foi estabelecida com sucesso com pelo menos um Client.

A barra de status sempre indica se houver algum erro e a ausência dele.

Também é mostrado para o usuário a quantidade de máquinas conectadas, os IPs respectivos e os nomes dos computadores. Os nomes dos computadores não precisa ser um nome técnico, pode ser definido pelo usuário do Client algum nome interno para facilitar a identificação. O usuário também poderá entrar na etapa de calibração ou parar a VI.

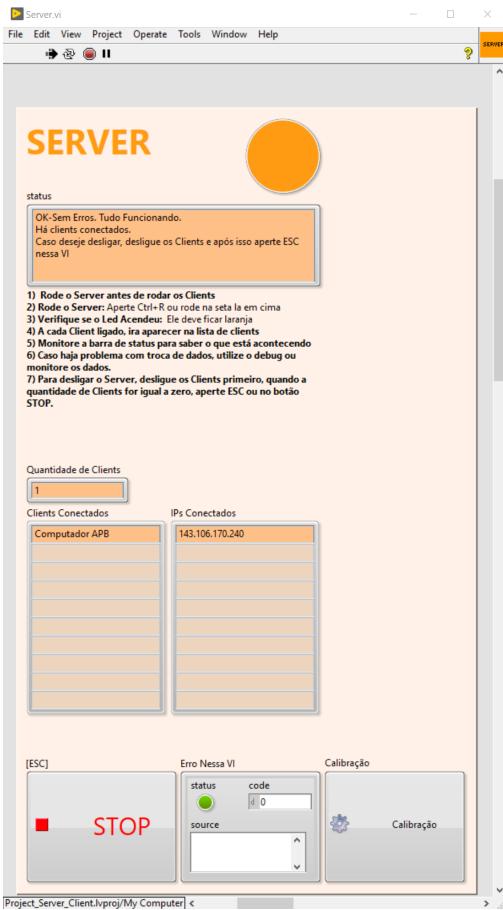


Figura 16: Interface do programa Server.vi

Fonte: Própria

5.2 Client

Essa VI segue também o mesmo padrão da interface Server, porém, diferente do Server ela pode ser rodada em segundo plano por uma VI Main com a aplicação do computador. Contudo, também foi desenvolvido uma interface intuitiva para caso precisar debugar o sistema, o usuário poderá abri-lá e confirmar as informações.

Ela segue o mesmo padrão de cores, porém com a cor Azul para representar o Client. O LED azul aceso indica que o sistema está funcionando e a comunicação foi estabelecida com sucesso.

O status de comunicação ajuda o usuário no que ele precisa fazer e se aparecer algum erro, qual providência tomar.

Um erro bastante comum que foi percebido durante o comissionamento, é rodar a VI Client antes do Server estar rodando, isso sempre irá gerar um erro pois o Server sempre precisa rodar antes pois é ele quem cria o escutador, e quando o programa Client tentar se conectar, não irá conseguir, porém, por se tratar de um erro bem comum e fácil de ser removido, esse padrão de erro é reconhecido pelo sistema e indica ao usuário que o Server está desligado e que precisa ser inicializado.

Uma coisa importante que vale ressaltar é que o Client sempre precisa saber qual a máquina Server, assim, o IP 143.106.170.240 foi alocado como uma constante, pois até então, essa máquina tinha sido estabelecida como a máquina Server do laboratório, caso a máquina Server troque de IP, é necessário alterar isso no código do Client.

No programa Client também é definido o tipo de dado que o usuário quer ler, como foi explicado na etapa de desenvolvimento.

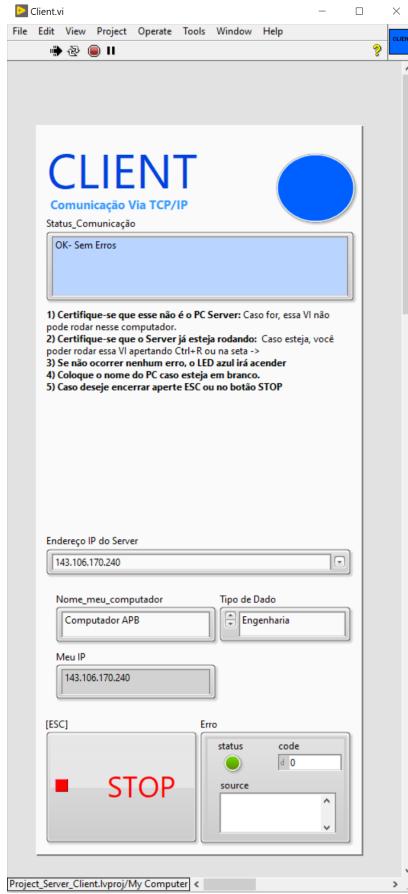


Figura 17: Interface do programa Client.vi

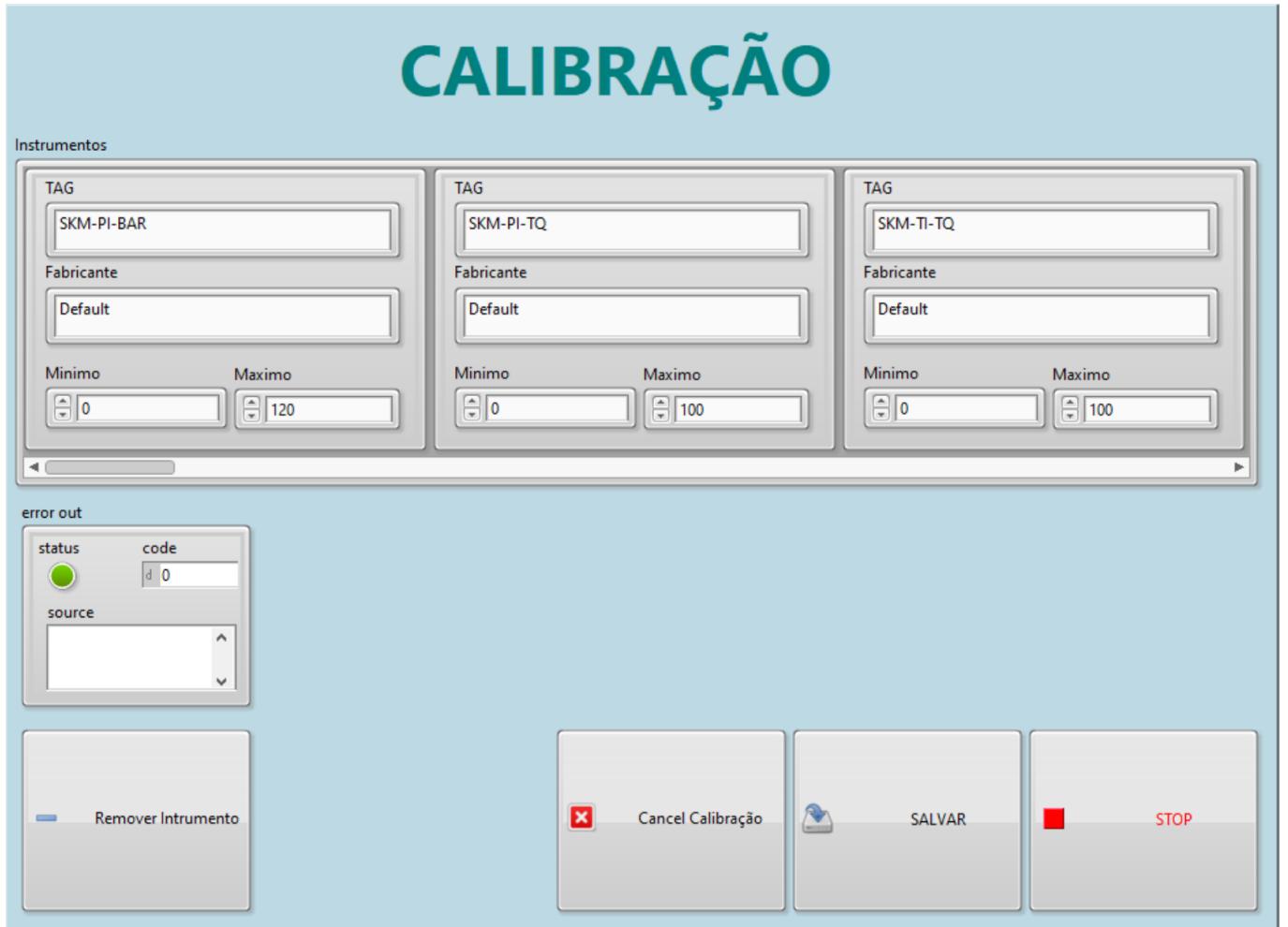
Fonte: Própria

5.3 Interface de Calibração

A interface de calibração foi pensada para que seja simples e eficiente ao msm tempo, assim sendo, a ideia é nunca o usuário ou o desenvolvedor precisar editá-la, adicionar ou remover instrumentos. Dessa forma, o display que mostra é um array de cluster com um barra de slide, assim se instrumentos forem removidos/adicionados, o próprio array se adequará. Como pode ser visto na abaixo, o usuário pode procurar o instrumento desejado e alterar as informações pertinentes, também, caso ele desejar, pode cancelar todo o processo e os novos valores não serão salvos.

Entetanto, caso o usuário tiver certeza do que está fazendo, poderá salvar os novos valores.

A interface de comunicação foi testada com um usuário que não participou do desenvolvimento e o mesmo conseguiu operar com êxito e entendeu as funções de cada botão. O programa demonstrou-se robusto e não apresentou erros.



Fonte: Própria

5.4 Testando a comunicação

Nos diversos testes feitos, o sistema se comportou robustamente, e a troca de dados foi feita com sucesso.

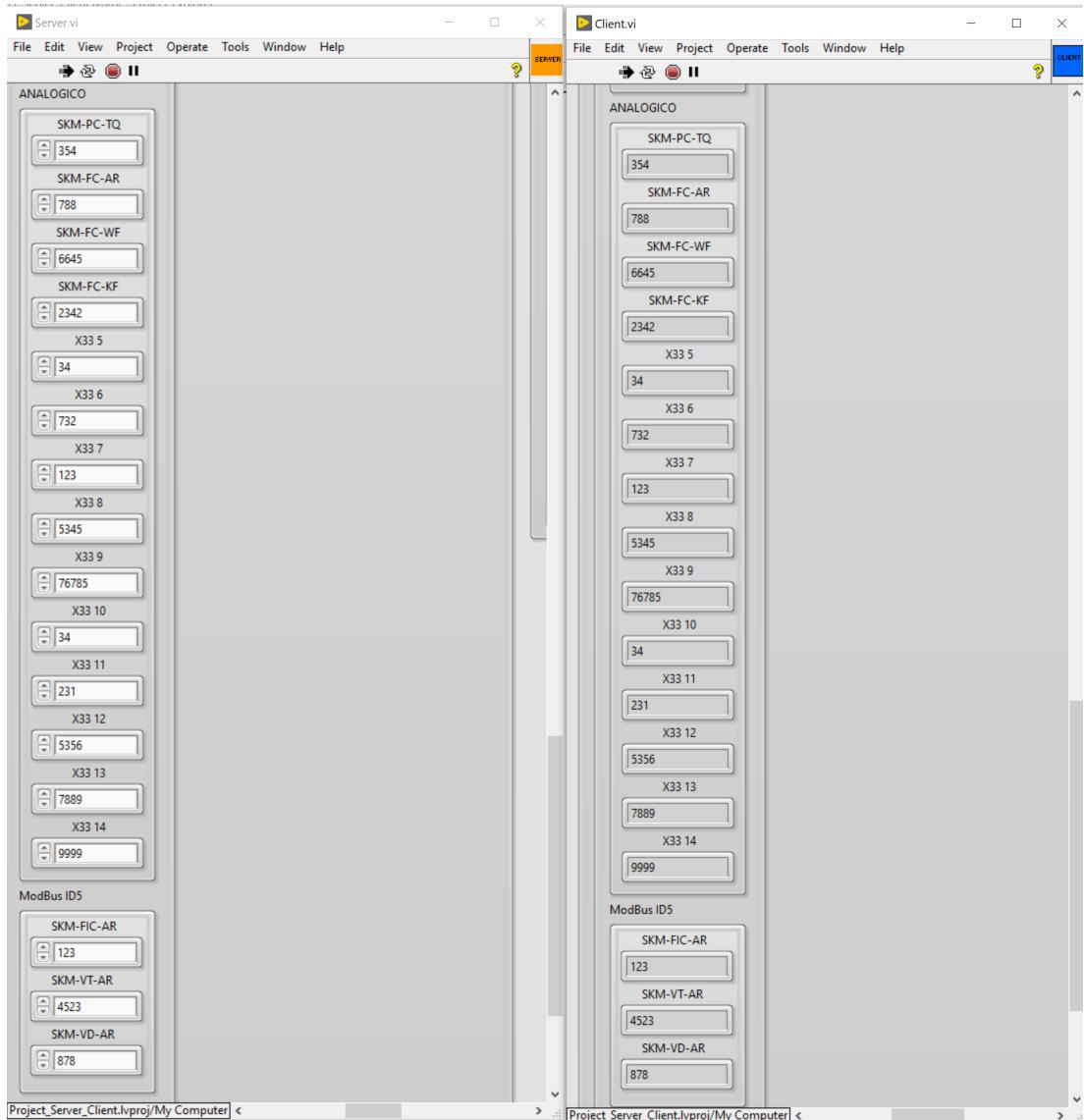


Figura 19: Troca de dados entre Server e Client via TCP/IP
Fonte: Própria

Foram feitos testes em casos mais extremos:

1. Desligando o Client e ligando várias vezes em sequência
2. Múltiplos Clients abertos em diferentes máquinas e na mesma máquina
3. Parada de funcionamento forçadamente do Server e reconexão minutos depois, o sistema voltou-se a comunicar.
4. Deixou-se o sistema rodando por 4 horas sem interrupção e o sistema demonstrou-se robusto.

6 Conclusão

Apesar dos diversos resultados positivos, o sistema ainda necessita de um maior desenvolvimento para ser aplicado ao caso real do laboratório.

No caso dos múltiplos Clients rodando ao mesmo tempo, a ideia é sempre um client ser responsável por variáveis diferentes então não é pra existir conflitos de valores para a entrada do CLP, porém, não podemos garantir que o usuário possa se confundir e escrever em variáveis equivocadas, assim, é necessário o desenvolvimento de um sistema mais robusto para impedir múltiplas escritas na mesma variável.

Apesar de termos um arquivo de configuração, todo instrumento com uma curva de medida necessita do Certificado de Calibração que comprova a calibração. Deixar o certificado vinculado ao instrumento pelo sistema seria uma ideia muito útil que ajudaria na profissionalidade técnica do laboratório e o sistema Server deve ter um mecanismo de disponibilização desses certificados para o operador.

Também, apesar do programa ter sido testado com conjuntos de variáveis menores do que a realidade vai exigir, é necessário dar prosseguimento com o sistema e incorporá-lo no laboratório. Diversos testes devem ser feitos para conferir a robustez do sistema.

7 Referências

Labone; Taubaté,SP;<<https://www.laboneconsultoria.com.br/labview/>>; Acesso em 09/02/2023.

BasuMallick, Chiradeep;18/042022;<<https://www.spiceworks.com/tech/networking/articles/tcp-vs-udp>>;Acesso em 09/02/2023.