INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
DO RIO GRANDE DO NORTE
CAMPUS CAICÓ

HUGO RAFAEL DE MEDEIROS FERNANDES
JOSÉ GERALDO NUNES NETO

APLICAÇÃO DO PROTOCOLO DE COMUNICAÇÃO MODBUS NO CONTROLE DO BOMBEAMENTO DE ÁGUA PARA ESTAÇÕES ELEVADAS

CAICÓ - RN
2016

HUGO RAFAEL DE MEDEIROS FERNANDES
JOSÉ GERALDO NUNES NETO

APLICAÇÃO DO PROTOCOLO DE COMUNICAÇÃO MODBUS NO CONTROLE DO BOMBEAMENTO DE ÁGUA PARA ESTAÇÕES ELEVADAS

Trabalho de Conclusão de Curso apresentado ao Curso Técnico em Eletrotécnica do Instituto Federal de Educação, Ciência e Tecnologia do Rio Grande do Norte, em cumprimento às exigências legais como requisito parcial à obtenção do título de Técnico em Eletrotécnica.

Orientador: Ms. Francisco das Chagas Souza Júnior

CAICÓ - RN
2016

HUGO RAFAEL DE MEDEIROS FERNANDES
JOSÉ GERALDO NUNES NETO

APLICAÇÃO DO PROTOCOLO DE COMUNICAÇÃO MODBUS NO CONTROLE DO BOMBEAMENTO DE ÁGUA PARA ESTAÇÕES ELEVADAS

Trabalho de Conclusão de Curso apresentado ao Curso Técnico em Eletrotécnica do Instituto Federal de Educação, Ciência e Tecnologia do Rio Grande do Norte, em cumprimento às exigências legais como requisito parcial à obtenção do título de Técnico em Eletrotécnica.

Orientador: Ms. Francisco das Chagas Souza Júnior

Aprovado em: 08/11/2016

Banca Examinadora

Ms. Francisco das Chagas Souza Júnior - Orientador
Instituto Federal de Educação, Ciência e Tecnologia do Rio Grande do Norte

Giancarlos Costa Barbosa - Examinador
Instituto Federal de Educação, Ciência e Tecnologia do Rio Grande do Norte

Jonas Damasceno Batista de Araújo - Examinador
Instituto Federal de Educação, Ciência e Tecnologia do Rio Grande do Norte

F363a Fernandes, Hugo Rafael de Medeiros.
Aplicação de protocolo de comunicação modbus no controle do bombeamento de água para estações elevadas / Hugo Rafael de Medeiros Fernandes, José Geraldo Nunes Neto– Caicó: 2016.
69 f. il.

Trabalho de Conclusão do Curso de Eletrotécnica (Curso Técnico em Eletrotécnica) - Instituto Federal de Educação, Ciência e Tecnologia do Rio Grande do Norte, Caicó, 2016.

Orientador: Me. Francisco das Chagas Souza Júnior

1. Arduino. 2. Água. 3. Eletrotécnica.
I. Nunes Neto, José Geraldo. II. Título.

CDU 621

Ficha elaborada pela Biblioteca do IFRN/ *Campus* Caicó.

Dedico este trabalho a minha mãe, que esteve sempre presente em todos os momentos da minha vida. À nossa família e amigos, que nos incentivaram e apoiaram em cada momento, servindo de base para a conquista da primeira de muitas vitórias que ainda virão.

AGRADECIMENTOS

A todos os professores do Instituto Federal de Educação, Ciência e Tecnologia do Rio Grande do Norte (IFRN) que colaboraram e construíram bases sólidas no nosso desenvolvimento e aprendizagem para o crescimento profissional. Seus nomes são inesquecíveis e por isso, dedicamos-lhes nossa profunda admiração e respeito.

RESUMO

A água, como a eletricidade, é um fator de saúde e de progresso para as comunidades, principalmente aquelas situadas em locais remotos. É um dos fatores que permitem a fixação do homem no campo, evitando seu deslocamento para as cidades com todos os problemas decorrentes. Em muitas regiões do Brasil, principalmente no Nordeste, populações inteiras vivem com uma grande carência de água, impedindo seu desenvolvimento econômico e mantendo sua condição de miséria e de doença. Na época de seca, rios e córregos secam, as fontes ficam distantes e essas populações têm que percorrer grandes distâncias para buscar água para o próprio consumo, muitas vezes água de baixa qualidade Frequentemente, essas populações estão localizadas sobre reservatórios de água de alta qualidade, situados nos lençóis subterrâneos. A falta de conhecimentos e de recursos financeiros, dificulta o acesso a essa água. Atualmente, as alternativas tecnológicas para bombeamento autónomo de água possuem elevado custo no mercado e de difícil implantação pelos usuários. Complicando mais ainda essa tarefa. Este trabalho busca aplicar o protocolo de comunicação de dados ModBus junto a plataforma de desenvolvimento Arduino, para a elaboração de um sistema autónomo de bombeamento de água para estações elevatórias de baixo custo.

Palavras-chave: Arduino. ScadaBR. Protocolo ModBus. Bombeamento de Água.

ABSTRACT

Water, like electricity, is a health factor and progress for communities, especially those situated in remote locations. It is one of the factors that allow the attachment of the people in the countryside, avoiding its displacement to the cities with all the problems arising. In many regions of Brazil, mainly in the Northeast, whole peoples are living with a severe shortage of water, impeding their economic development and maintaining its condition of poverty and disease. During the dry season, rivers and streams dry up, water supplies are distant and these people have to travel long distances to fetch water for their own consumption, often of low quality water. Often, these populations are located on high quality water reservoirs, located in groundwater. The lack of knowledge and financial resources hampers their access to that water. Currently, alternative technologies for autonomous pumping water have high cost in the market and difficult to implement by users. Further complicating this task. This paper seeks to apply the ModBus data communications protocol with the Arduino development platform for the establishment of an autonomous water pumping system for low-cost pumping stations.

Keywords: Arduino. ScadaBR. Protocol ModBus. Water Pumping

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE CÓDIGOS

SUMÁRIO

1. INTRODUÇÃO

A necessidade de atender demandas cada vez maiores em espaços de tempo cada vez mais reduzidos, fez com que a automação industrial ganhasse força dentro das linhas de produção. A substituição da mão-de-obra humana por robôs torna a produção mais rápida e eficiente, havendo assim menos desperdício de insumos além de ser possível programar o fluxo de produção de acordo com a demanda do momento.

Nos últimos anos, o setor de automação tem evoluído bastante, diversas soluções estão no mercado com custos variados para uma infinidade de aplicações. No entanto, em muitos casos soluções padronizadas e comerciais tem elevado custo de projeto, sendo também de difícil implementação, fazendo com que várias pessoas percam o interesse em investir nessa área.

Indo em contra partida, muitas empresas e academias, tem estudado e investido na busca de soluções mais baratas para resolver problemas de automação. Dessa forma, tecnologias Open Source (Código Aberto) que não possuem licenças envolvidas para sua utilização, tem ganhado espaço e tornado mais acessível o desenvolvimento de tecnologias por indivíduos dos mais variados graus de escolaridade e renda.

Atualmente, projetos de automação industrial vem se popularizando. Tendo em vista que, pessoas comuns e muitas vezes com pouco conhecimento na área, são capazes de realizar simples aplicações que contribuem no seu cotidiano, como também para divulgação do setor, fortalecendo assim, a busca desse tipo de tecnologia por parte de outras pessoas e o interesse em cursos e capacitações nessa área.

Por fim, a automação industrial é fundamental no mercado contemporâneo, em especial com o objetivo de criar uma vantagem competitiva que permita a sobrevivência do empreendimento em mercados acirrados. A possibilidade de flexibilizar a produção permite que a empresa possa explorar nichos de mercados diferentes do comum, aumentando assim a sua parcela de consumidores.

1.1 Contextualização

Nesse trabalho será pesquisado o protocolo de comunicação de dados ModBus, seu funcionamento e implementação. O problema proposto aqui, é a utilização do ModBus em um projeto automatizado de bombeamento de água para estações elevadas. A motivação que levou para o desenvolvimento desse trabalho, foi a busca por alternativas de baixo custo na implementação de projetos de automação industrial.

1.2 Objetivo

Esse trabalho tem como objetivo a utilização do protocolo de comunicação de dados ModBus, para automação de uma aplicação, mostrando assim, que se pode desenvolver projetos de automação industrial a partir de ferramentas gratuitas e de fácil implementação.

1.3 Justificativa

- Adquirir conhecimentos na área.
- Tornar acessível para sociedade soluções em automação industrial, com baixo custo e de fácil desenvolvimento.
- Mostrar a aplicação de conceitos desenvolvidos na academia

1.4 Método de pesquisa

A forma com que foi desenvolvida a pesquisa desde trabalho, foi utilizando bibliografias, trabalhos relacionados e testes em laboratório.

1.5 Estrutura do trabalho

Nessa parte, destacamos superficialmente as partes que integram este trabalho. Basicamente, o texto está dividido em duas partes, a primeira contempla os capítulos 1, 2 e 3, onde buscamos explicar de forma detalhada as ferramentas utilizadas no projeto. A segunda parte, que é composta do capitulo 4, mostra o

processo de implementação das ferramentas vistas nos capítulos anteriores, para o desenvolvimento de um projeto de automação industrial. No Capitulo 1, é mostrado os detalhes da plataforma de desenvolvimento Arduino, suas características e funcionamento. No Capitulo 2, veremos os conceitos em torno do Protocolo de Comunicação de Dados ModBus. No Capitulo 3, será apresentado o Software ScadaBR. Finalmente no Capitulo 4, serão mostrados os passos para o desenvolvimento do projeto de automação, que visa, o transporte autônomo de água para estações elevadas.

2. REFERENCIAL TEÓRICO

O propósito do referencial teórico é levantar as várias categorias que serão utilizadas para dar conta dos fenômenos a serem abordados e explicados, nos quais se apoia para conduzir o trabalho investigativo e o raciocínio. Sendo assim, de vital importância para a pesquisa.

Nesta parte, introduzimos os conceitos básicos que permitiram a realização do projeto prático desse trabalho. Aqui serão apresentados três capítulos que explicaram de forma clara as principais tecnologias utilizadas no projeto.

Os capítulos a seguir estão estruturados como é mostrado:

- Capitulo 1 - Arduino
- Capitulo 2 - ModBus
- Capitulo 3 - ScadaBR

Vale aqui ser dito, que os leitores que já possuírem entendimento nos assuntos acima citados, podem seguir para o capitulo 4, onde é discutido a aplicação implementada neste trabalho.

Capitulo 1: Arduino

O Arduino é uma plataforma de prototipagem eletrônica de hardware livre, seu projeto iniciou na cidade de Ivrea, Itália, em 2005, com a finalidade de integrar projetos escolares de forma a ter um orçamento menor que outros sistemas de prototipagem disponíveis na época. Em 2006, recebeu uma menção honrosa na categoria Comunidades Digitais, pela Prix Ars Electronica e em outubro do ano de 2008 atingiu a marca de mais de 50.000 placas vendidas. Atualmente, surgiram vários projetos paralelos que se inspiraram no arduino, criando diversas cópias que muitas vezes acabam recebendo seus próprios nomes.

Seu hardware é feito através de um microcontrolador Atmel AVR, apesar de não ser um requisito formal. Possui diversos acessos de entrada e saída de informações, esses com a ajuda de sensores e atuadores, permitem a construção de aplicações que possam receber parâmetros físicos e responder a isso. Muitas das funções do arduino, se devem a bibliotecas de códigos que simplificam sua programação, escritas num sintaxe muito similar as linguagens C/C++.

O usuário tem a possibilidade de montar o seu próprio sistema, entretanto os mantenedores oferecem um serviço de venda do produto pré-montado, e também por distribuidores oficiais com pontos de venda mundiais. No entanto, um dos fatores determinantes para a enorme versatilidade e popularidade da plataforma Arduino são seus Shields (Módulos), que podem ser adquiridos de terceiros ou fabricados pessoalmente. Essas placas podem ser conectadas ao Arduino, expandindo suas capacidades, possibilitando uma infinidade de aplicações de maneira simples e rápida. [MONK, 2014]

O Arduino foi projetado com a finalidade de proporcionar aos seus usuários um fácil entendimento, de programação e aplicação, além de poder ser utilizado em multiplataforma, o que possibilita a sua integração e configurado em ambientes Linux, Mac OS e Windows. Assim, um dos pontos fortes e também o grande diferencial deste dispositivo, é o fato do projeto ser open-source, a qual possibilita o desenvolvimento e divulgação gratuita de seus projetos em comunidades de projetistas espalhados por todo o mundo e para diversas plataformas. Atualmente no principal fórum de discussão sobre arduino, é encontrado mais de 2,5 Milhões de

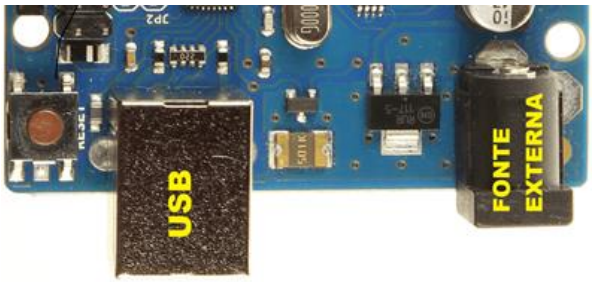
postagens e quase 400 Mil tópicos de discussão de usuários se envolvendo e contribuindo com a comunidade.

1. ARDUINO UNO

Existem várias versões do Arduino disponíveis no mercado, contudo a que foi utilizada na elaboração do projeto atual recebe o nome de UNO. A referida placa encontra-se atualmente em sua terceira revisão e no site do Arduino você pode baixar seu esquema elétrico em formato PDF, ou até mesmo todos os arquivos do projeto para edição. A seguir serão apresentadas as principais características do seu hardware.

A alimentação da placa poderá ser feito através de uma conexão USB ou por uma fonte de alimentação externa, como exibido na figura 1.

Figura 1 - Alimentação da placa



Fonte: http://www.embarcados.com.br/arduino-uno/

Para a alimentação externa, os valores de tensão devem estar entre os limites de 7V a 20V, sendo um problema se alimentada com uma tensão abaixo de 7V, pois o dispositivo poderá apresentar alguma instabilidade da mesma forma que uma alimentação com tensão acima de 20V, poderá sobreaquecer e danificar a placa. Outra forma de alimentação é pela conexão USB, assim a tensão não precisa ser estabilizada pelo regulador de tensão.

No design da plataforma são encontrados diversos pinos que auxiliam na montagem dos circuitos. Esses pinos são referências aos mesmos encontrados no microcontrolador, estando conectados eletricamente. Na figura 2, é mostrado os

pinos correspondentes a função de alimentação. E na tabela 1, é mostrado suas respectivas funções.

Figura 2 - Pinos de alimentação

Fonte: http://www.embarcados.com.br/arduino-uno/

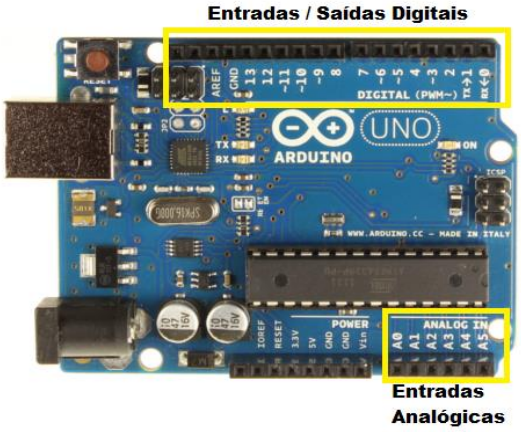
Tabela 1 - Características dos pinos de alimentação

PINOS	DESCRIÇÃO
IOREF	Fornece uma tensão de referência para que placas de expansão possam selecionar o tipo de interface apropriada, dessa forma, funcionando com placas que são alimentadas com 3,3V ou 5V.
RESET	Pino conectado ao RESET do microcontrolador. Inicializa as configurações da placa.
3,3V	Fornece tensão de 3,3V com corrente máxima de 50 mA.
5V	Fornece tensão de 5 V.
GND	Pino de referência (Terra).
VIN	Pino para alimentar a placa através de dispositivos ou baterias externas. Quando a placa é alimentada através externamente, a tensão da fonte estará nesse pino.

Fonte: Elaborado pelos autores, (2016).

A placa possui pinos de entrada e saídas digitais como também pinos de entradas e saídas analógicas localizados em suas bordas. Na figura 3, isso pode ser ilustrado.

Figura 3 - Pinos de entrada e saída



Fonte: http://www.embarcados.com.br/arduino-uno/

A placa Arduino UNO possui 14 pinos que podemos usar como entradas ou saídas digitais. Estes Pinos operam em 5 V, os quais podem fornecer ou receber uma corrente máxima de 40 mA. Em tabela 2, é mostrado as características dos pinos de entrada e saída.

Tabela 2 - Pinos de Entrada e Saída

Pinos	Descrição
A0, A1, A2, A3, A4, A5	Pinos analógicos. Sua função é ler ou escrever valores contínuos.
0 e 1 ou RX e TX	Pinos Digitais. Sua função é executar tarefas de transmissão e recepção de sinais.
2, 4, 7, 8, 12 e 13	Pinos Digitais. Leem e Escrevem sinais digitais.
3, 5, 6, 9, 10 e 11	Pinos Digitais. Realizam todas as funções típicas dos pinos digitais, entretanto, podem ativar a técnica PWM (Pulse-Width Modulation) que permite a manipulação de valores analógicos.

Fonte: Elaborado pelos autores, (2016).

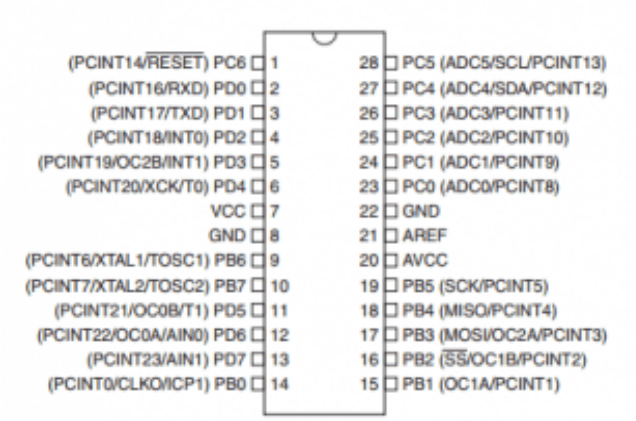
2. ATMEL

O Arduino UNO tem como componente principal o microcontrolador ATMEL ATmega328, um dispositivo de 8 bits da família AVR com arquitetura RISC (*Reduced Instruction Set Computer*). Ele conta com 32 KB de memória Flash onde o

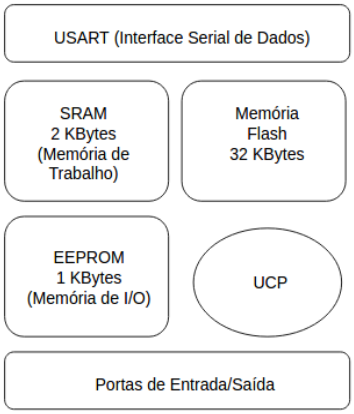
código é armazenado, 2 KB de SRAM usada como memória de trabalho e 1 KB de EEPROM servindo como memória de I/O. Opera numa frequência de até 20 MHz, valor esse gerado pelo cristal de quartzo instalado na placa, onde dita o clock usado pelo microcontrolador. Sua tensão de operação vária de 1,8 à 5,5 Volts. O dispositivo ainda contém um modulo USART para permitir a comunicação serial.

Especialmente o ATMEL ATmega328, possui 28 pinos, sendo que 23 desses podem ser utilizados como I/O. As figuras 4 e 5, mostram o esquema físico do chip e seu diagrama de blocos.

Figura 4 - Pinagem e Diagrama do ATmega328



Fonte: http://www.circuitstoday.com/avr-atmega8-microcontroller-an-introduction



Fonte: Elaborado pelos autores, (2016).

3. ESTRUTURA DO CÓDIGO ARDUINO

O código 1, mostra um exemplo de uma aplicação arduino que acende e apaga um LED a cada um segundo, ou seja, piscando a cada 1 segundo. Por esse exemplo, podemos ver o funcionamento da estrutura básica dos códigos arduino. Todos os códigos arduino possuem basicamente as funções “Setup” e “Loop”, a primeira busca servir de mecanismo de configuração da plataforma, no exemplo vemos na linha 2 que o pino digital 13 está sendo configurado como porta de saída (OUTPUT) pela função “pinMode”. Já a segunda, serve para executar as instruções repetidas vezes. Na linha 5, a função “digitalWrite” escreve o valor lógico alto (HIGH) para o pino 13, em seguida o programa espera 1000 milissegundos (1 segundo)

antes de executar a instrução seguinte. Na linha 7, o processo é o mesmo que antes, com a diferença do valor lógico ser mudado para baixo (LOW). Finalmente a função Loop retorna para linha 5 e repete as instruções em seu escopo.

Código 1 - Exemplo da estrutura de um código de Arduíno

```
1  void setup() {
2          pinMode(13, OUTPUT);
3  }
4  void loop() {
5          digitalWrite(13, HIGH);
6          delay(1000);
7          digitalWrite(13, LOW);
8          delay(1000);
9  }
```

Fonte: Elaborado pelos autores, (2016).

Capitulo 2: Modbus

O protocolo ModBus é uma estrutura de mensagem desenvolvida pela Empresa Modicon em 1979 (atualmente mantido pela Modbus Organization desde 2004), usada para estabelecer comunicação entre dispositivos organizados como Mestre/Escravo ou Cliente/Servidor. O protocolo é bastante utilizado em sistemas de automação industrial, por ser uma das soluções de redes mais barata, ser de fácil aplicação, com capacidade de se adequar facilmente a diversos meios físicos e por ser um padrão de comunicação de dados sob licença livre, faz do ModBus ser um protocolo bastante popular e amplamente implementado no setor industrial para aquisição de sinais e comandar atuadores.

É desenvolvido para implementar a técnica conhecida como Mestre/Escravo (ou no inglês, Master/Slave), onde somente o dispositivo Mestre pode iniciar as transações (chamadas de Queries). Os dispositivos escravos respondem de acordo com o Mestre, ou de acordo com a tarefa em questão.

O protocolo ModBus está na camada de rede de aplicação, e utiliza interfaces RS-232, Ethernet ou RS-485, como meios físicos (Camada de Enlace), onde possui comandos para envio de dados discretos (digitais) ou numéricos (Analógicos). A velocidade de comunicação varia em cada um desses padrões, bem como o comprimento máximo da rede e o número máximo de dispositivos conectados.

Dependendo do meio pelo qual o protocolo será implementado, o ModBus especifica a forma que a mensagem será enviada, empacotada e descompactada. Na prática, se encontra dois padrões por onde uma mensagem ModBus pode ser transmitida, o Serial e o Ethernet.

O padrão de transmissão Serial utiliza interfaces RS-232 e RS-485, esses com características próprias, mais muito utilizados no setor industrial.

A interface RS-232 (Recommendad Standart-232) ou EIA-232 (Electronic Industries Alliance-232) é utilizado apenas em comunicações do tipo ponto a ponto, ou seja, só admite dois dispositivos na rede, que no caso do protocolo Modbus representa o mestre e um escravo. A velocidade máxima desse padrão está em torno de 115Kbps, mas em alguns casos podem ser encontradas taxas um pouco maiores, a distância máxima entre os dispositivos da rede está em torno de 30 Metros.

Já a interface RS-485 (Recommendad Standart-485) ou EIA-485 (Electronic Industries Alliance-485) é muito utilizado na indústria e sem dúvida é um dos padrões mais utilizados pelo protocolo Modbus. Esse padrão permite trabalhar com taxas de comunicação que podem chegar a 12Mbps e em alguns casos até 50Mbps, vale lembrar que quanto maior o comprimento da rede menor será a velocidade de comunicação, a distância máxima da rede está em torno de 1,2 km, e o número máximo de dispositivos no barramento da rede é de 32.

No padrão de transmissão Ethernet, os dados são representados em formato binário e encapsulados em pacotes TCP, isso permite a utilização do meio físico Ethernet (IEEE 802.3). Este possui algumas variações, podendo chegar a 100Mbps ou até 10Gbps. A distância máxima pode variar de 100 Metros até próximo de 200 Metros dependendo do tipo de cabo utilizado e das condições de instalação do mesmo. Em alguns casos é possível utilizar redes em fibra ótica, fato que permite alcançar distâncias maiores e melhores taxas de comunicação, bem como utilizar comunicação Wireless.

Uma das maiores vantagens da utilização do padrão Ethernet, é as aplicações envolvendo o uso da Internet, tendo em vista que, alguns processos podem ser monitorados ou até controlados remotamente por esses tipos de redes. Vantagens essas que em muitos casos as redes Serial não possuem, sendo quase que exclusivamente acessadas localmente.

Entretanto, redes Ethernet não se adaptam muito bem a equipamentos antigos, pois muitos não possuem suporte a essa tecnologia. Dispositivos antigos que só utilizam meios Serial, são um exemplo dessa falta de compatibilidade. No entanto, existem no mercado diversos conversores capazes de fornecer suporte a equipamentos antigos, mais nem sempre é possível.

1. ESTRUTURA DE UMA MENSAGEM MODBUS

Uma mensagem ModBus é usada para permitir a comunicação do mestre da rede com os dispositivos escravos. Na tabela 3, é mostrado um exemplo de mensagem ModBus.

Tabela 3 - Estrutura de uma mensagem ModBus

Endereço	Função	Endereço do Registrador		Quantidade de Registradores	
11_{16}	03_{16}	00_{16}	$6B_{16}$	00_{16}	03_{16}

Fonte: http://www.simplymodbus.ca/

Basicamente, uma mensagem ModBus está representada em hexadecimal onde contém o endereço do escravo ($11_{16} = 17_{10}$), a função a ser executado ($03_{16} = 03_{10}$), o endereço do registrador que se quer acessar ($006B_{16} = 107_{10}$), e o número de registradores que serão retornados usando como referência o anteriormente citado ($0003_{16} = 03_{10}$). Assim nesta mensagem, o mestre da rede está solicitando a permissão de leitura (Função) a 3 (Quantidade de Registradores) registradores do dispositivo escravo 17 (Endereço). Ao final dessa requisição, o dispositivo escravo retornará os registradores 107 (Endereço do Registrador), 108, 109.

Entretanto, uma mensagem ModBus não é suficiente para o envio pela rede, pois é necessário adicionar campos para manter o controle e integridade dos dados. Dessa maneira, o processo que permite isso, é o chamado empacotamento. No ModBus, existem duas formas de empacotar uma mensagem, pelo modo ASCII ou pelo modo RTU.

- Modo ASCII

No modo ASCII (*American Standard Code for Information Interchange*), a estrutura usa caracteres legíveis para representar o pacote. Por ser legível, acaba gastando mais tempo no processo de codificação e empacotamento, tornando sua transmissão mais lenta.

Nessa estrutura as informações são codificados através da tabela ASCII, que usa 7 bits para representar um caractere. Além disso, são adicionados um bit de inicio (start bit), um bit de parada (stop bit) e um bit de paridade (parity bit), fazendo

o Byte ter tamanho de 10 bits. A tabela 4, mostra a representação de um Byte no modo ASCII.

Tabela 4 - Tamanho do Byte ASCII

Start Bit	Dados	Parity Bit	Stop Bit
1_2	$100\ 0001_2$	0_2	1_2

Fonte: http://www.simplymodbus.ca/

Outra particularidade do modo ASCII, é que todo campo da mensagem ModBus é convertido para caracteres ASCII, sendo necessário 2 caracteres ASCII para representar cada campo da mensagem ModBus. As tabelas 5 e 6, realiza a conversão para ASCII e expande a mensagem ModBus até sua representação binária.

Tabela 5 - Conversão da mensagem ModBus para ASCII

Endereço	Função	Endereço do Registrador		Quantidade de Registradores	
11_{16}	03_{16}	00_{16}	$6B_{16}$	00_{16}	03_{16}
$31\ 31_{ASCII}$	$30\ 33_{ASCII}$	$30\ 30_{ASCII}$	$36\ 42_{ASCII}$	$30\ 30_{ASCII}$	$30\ 33_{ASCII}$

Fonte: http://www.simplymodbus.ca/

Tabela 6 - Expansão do campo endereço da mensagem ModBus

Endereço							
11_{16}							
31_{ASCII}				31_{ASCII}			
1_2	$001\ 1111_2$	0_2	1_2	1_2	$001\ 1111_2$	0_2	1_2

Fonte: http://www.simplymodbus.ca/

Para finalizar o empacotamento da mensagem ModBus, são adicionados mais 3 componentes, um caractere “dois pontos” (:) no inicio da mensagem usado

para informar o inicio da mensagem, um caractere chamado LRC (*Longitudinal Redundancy Check*) usado para checar a integridade da mensagem e dois caracteres (CR e LF) usados para informar o final da mensagem. A tabela 7, mostra o resultado final do empacotamento da mensagem ModBus.

Tabela 7 - Mensagem ModBus empacotada no modo ASCII

Inicio	End.	Função	Endereço do Registrador		Quantidade de Registradores		LRC	CR	LF
:	11_{16}	03_{16}	00_{16}	$6B_{16}$	00_{16}	03_{16}	$7E_{16}$	/r	/n
$3A_{ASCII}$	3131 ASCII	3033_{ASCII}	3030_{ASCII}	3642_{ASCII}	3030_{ASCII}	3033 ASCII	3745_{ASCII}	$0D_{ASCII}$	$0A_{ASCII}$

Fonte: http://www.simplymodbus.ca/

Uma característica presente nos pacotes ASCII é a adição do campo LRC, permite se verificar a integridade do pacote. O processo consiste na realização da soma aritmética da mensagem ModBus, em seguida é aplicado a função Complemento de Dois (Está função encontra o complemento, ou a negação de um valor) no resultado. A cada pacote enviado é verificado pelo destinatário, se a soma da mensagem é complemento do LRC. Se esse for diferente, significa que a mensagem sofreu algum tipo de alteração no transporte, perdendo sua integridade e devendo ser ignorada. Esse processo de geração do LRC é detalhado na tabela 8.

Tabela 8 - Processo de geração do LRC

17_{10}	11_{16}	$0001\ 0001_2$
3_{10}	03_{16}	$0000\ 0011_2$

	0_{10}		00_{16}		$0000\ 0000_2$
	107_{10}		$6B_{16}$		$0110\ 1011_2$
+	0_{10}	+	00_{16}	+	$0000\ 0000_2$
	3_{10}		03_{16}		$0000\ 0011_2$
	130_{10}		82_{16}		$1000\ 0010_2$
LRC	-130_{10}	LRC	$7E_{16}$	LRC	$0111\ 1110_2$

Fonte: http://www.simplymodbus.ca/

O pacote ilustrado na tabela 7, representa o estado final de uma requisição ModBus no modo ASCII. Seu tamanho possui 17 Bytes (170 bits).

- Modo RTU

No modo RTU (*Remote Terminal Unit*), os dados são representados na forma hexadecimal. Diferente do modo ASCII, o RTU não utiliza nenhum método de codificação dos dados, tornando o processo bem mais rápido. Outra característica do RTU, é seu tamanho de Byte, que possui 8 bits para representar informações, em comparação aos 7 bits do modo ASCII, permitindo assim, o envio de mais dados por Byte. O tamanho de cada Byte RTU é mostrado na tabela 9.

Tabela 9 - Tamanho do Byte RTU

Start Bit	Dados	Stop Bit
1_2	$0100\ 0001_2$	1_2

Fonte: http://www.simplymodbus.ca/

O Byte RTU, possui 8 bits para representar os dados, um bit de inicio (Start Bit) e um bit de parada (Stop Bit) que delimitam o começo e fim do Byte. No total cada Byte RTU contém 10 bits para transmitir informações.

O processo de empacotamento do modo RTU, é adicionado um campo chamado de CRC (*Cyclic redundancy check*) no final do pacote. Sua função é similiar ao LRC do modo ASCII, sendo um verificador de integridade do pacote.

Tabela 10 - Pacote RTU

Endereço	Função	Endereço do Registrador		Quantidade de Registradores		CRC	
11_{16}	03_{16}	00_{16}	$6B_{16}$	00_{16}	03_{16}	76_{16}	87_{16}

Fonte: http://www.simplymodbus.ca/

O campo CRC é um método para identificação de erros, que se baseia em tratar sequências de bits transformando-as em polinómios, afim de aplicar um polinómio gerador para se determinar o resíduo da divisão dos dois polinómios e verificar o resultado. Se o resultado for igual aos anteriores a mensagem está correta.

Para melhor entendimento, na tabela 11 o campo endereço é expandido até sua representação binária. Assim, podemos determinar o tamanho do pacote RTU em 8 Bytes (80 bits).

Tabela 11 - Expansão do campo endereço do pacote RTU

Endereço		
11_{16}		
1_2	$0001\ 0001_2$	1_2

Fonte: http://www.simplymodbus.ca/

Algo que se pode verificar em um pacote RTU é que o mesmo não possui campos destinados a informar o inicio e final do pacote. No entanto, os dispositivos

utilizam uma métrica baseada em tempo para ditar o começo e fim de um pacote RTU, permitindo dessa forma, a comunicação.

2. FUNCIONAMENTO DA REDE

Em uma rede ModBus todos os dispositivos monitoram o inicio da mensagem enviada pelo mestre. No modo ASCII é verificado o caractere “:” para dar inicio ao processo de decodificação do pacote, em redes RTU o primeiro dado recebido já é o endereço do escravo que se pretende manter comunicação. Nos dois casos o dispositivo verifica se o endereço corresponde ao seu, caso contrário ignora a mensagem.

Existem três faixas de endereços possíveis nas redes ModBus, o valor zero que é o endereço de Broadcast, onde só é usado quando o mestre pretende mandar mensagens para todos os escravos da rede. Os valores de 1 até 247, são faixas de endereços atribuídas aos escravos. E a faixa de 248 à 255, são valores reservados. Dessa forma, uma rede ModBus pode manter até 256 endereços.

O mestre da rede ModBus não possui endereço, assim, sempre que um dispositivo escravo deseja responder a uma requisição do mestre, ele a empacota utilizando seu próprio endereço. O mestre monitora um pacote com o mesmo endereço da requisição e entende que se trata da resposta do dispositivo.

Logo em seguida, o próximo campo verificado é o de função. Este atributo dita a ação que será executada. No ModBus, existem diversos códigos que determinam a função. Na tabela 12, são mostrados alguns desses códigos.

Tabela 12 - Funções do Protocolo ModBus

Código da função	Descrição
1	Leitura de bloco de bits do tipo coil (saída discreta).
2	Leitura de bloco de bits do tipo entradas discretas.
3	Leitura de bloco de registradores do tipo holding.
4	Leitura de bloco de registradores do tipo input.
5	Escrita em um único bit do tipo coil (saída discreta).

6	Escrita em um único registrador do tipo holding.
7	Ler o conteúdo de 8 estados de exceção.
8	Prover uma série de testes para verificação da comunicação e erros internos.
11	Obter o contador de eventos.
12	Obter um relatório de eventos.
15	Escrita em bloco de bits do tipo coil (saída discreta).
16	Escrita em bloco de registradores do tipo holding.
17	Ler algumas informações do dispositivo.
20	Ler informações de um arquivo.
21	Escrever informações em um arquivo.

Fonte: http://www.simplymodbus.ca/

Para alguns comandos de diagnóstico, tais como reinício de comunicação, reset do módulo ou sincronização de relógio, podem ser utilizados comunicações do tipo broadcast, ou seja, destinada a todos os escravos simultaneamente.

No exemplo mostrado na tabela 3, o mestre faz uma requisição ao dispositivo escravo de endereço 11_{16} (17_{10}) e a mensagem pede acesso de leitura a 3 (0003_{16}) registradores começando do $006B_{16}$ (107_{10}). Na tabela 13, é mostrado a resposta do dispositivo escravo a essa requisição.

Tabela 13 - Resposta do Dispositivo Escravo

Endereço	Função	Tamanho do Retorno	Primeiro Registrador		Segundo Registrador		Terceiro Registrador	
11_{16}	03_{16}	06_{16}	00_{16}	$6B_{16}$	00_{16}	$6C_{16}$	00_{16}	$6D_{16}$

Fonte: http://www.simplymodbus.ca/

Capitulo 3: ScadaBR

A sigla SCADA é uma sigla do inglês para Supervisory Control And Data Acquisition, o que significa Controle Supervisório e Aquisição de Dados. Sistemas SCADA servem como interface entre o operador e processos dos mais variados tipos, como máquinas industriais, controladores automáticos e sensores dos mais variados tipos. Com sistemas SCADA são construídos desde aplicativos simples de sensoriamento e automação, até os famosos "Painéis de Controle" em empresas de geração e distribuição de energia elétrica, centrais de controle de tráfego e assim por diante.

Um SCADA típico deve oferecer drivers de comunicação com equipamentos, um sistema para registro contínuo de dados ("datalogger") e uma interface gráfica para usuário, conhecida como "IHM" ou Interface Homem-Máquina. Na IHM são disponibilizados elementos gráficos como botões, ícones e displays, representando o processo real que está sendo monitorado ou controlado. Entre algumas das funções mais utilizadas em sistemas SCADA estão:

- Geração de gráficos e relatórios com o histórico do processo;
- Detecção de alarmes e registro de eventos em sistemas automatizados;
- Controle de processos incluindo envio remoto de parâmetros, acionamento e comando de equipamentos;
- Uso de linguagens de script para desenvolvimento de lógicas de automação.

O software ScadaBR é desenvolvido em modelo Open-source, possuindo licença gratuita. Toda a documentação e o código-fonte do sistema estão à disposição, inclusive sendo permitido modificar e redistribuir o software se necessário.

Nasceu na MCA Sistemas em Florianópolis/SC, sendo ativamente desenvolvido na Fundação CERTI por uma equipe dedicada de profissionais, em parceria com mais duas empresas da região, a Unis Sistemas e a Conetec. Atualmente, a MCA Sistemas representa e distribui o software, oferecendo serviços de Projeto de Sistemas, Instalação, Configuração e Personalização do ScadaBR.

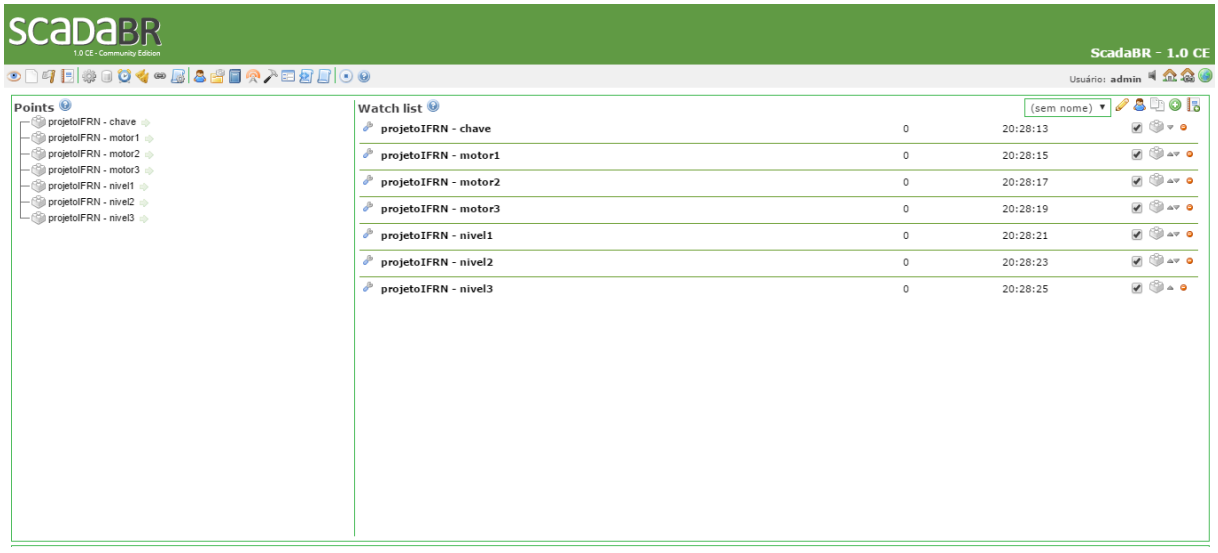
Completando 5 anos desde a primeira versão, o ScadaBR conta com mais de 70 mil downloads, com vários casos de uso nas áreas de automação de processos industriais, redes de distribuição (Água & Energia), automação predial e residencial, e aplicações de sensoriamento diversas. O ScadaBR possui suporte para mais de 20 protocolos de comunicação, sendo compatível com hardwares de centenas de fabricantes em todo o mundo.

1. VISÃO GERAL

O ScadaBR é uma aplicação Web multiplataforma baseada em Java, ou seja, qualquer equipamento rodando uma JVM (Máquina Virtual Java) podem executar o software a partir de um navegador de internet, sendo necessário também a instalação de um servidor Web para a comunicação, onde o Apache Tomcat é a escolha padrão.

Suas funcionalidades são acessadas pelos ícones no cabeçalho abaixo da logomarca do projeto. Quando o cursor do mouse pairar sobre um ícone, será exibida um balão de texto com uma descrição resumida de seu funcionamento. Além dos ícones de controle, no lado direito do cabeçalho é mostrado o nome do usuário que está logado no sistema e uma bandeira de alertas logo ao centro.

Figura 5 - Tela inicial do ScadaBR



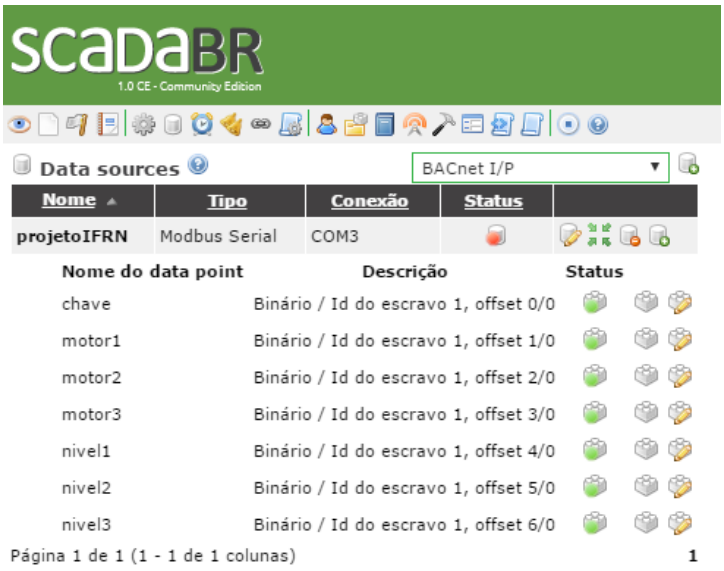
Fonte: Print Screen do Software ScadaBR

2. DATASOURCES E DATAPOINTS

O ScadaBR é uma aplicação que permite a integração com diversas tecnologia, sendo uma delas o protocolo ModBus. O mecanismo que permite essa integração é chamado de Datasource (Fonte de Dados). Um datasource contém todas as configurações da tecnologia que pretende-se utilizar, entretanto, para manipular os dados é necessário componentes chamados de Datapoints (Pontos de Dados). Assim, um datapoint sempre está associado a um datasource, podendo esse ter vários sobre o seu controle.

Os datapoints podem ser associados a dispositivos para exercerem tarefas de monitoramento e controle, como por exemplo, um sensor acompanhando a temperatura de um quarto, enquanto outro é capaz de acionar um atuador. Os dados trocados entre dispositivos e datapoints podem ser de tipos digitais, analógicos, multiestados ou alfanuméricos.

Figura 6 - Datasources e Datapoints



Fonte: Print Screen do Software ScadaBR

3. RECURSOS DO SCADABR

O usuário dependendo da aplicação, pode utilizar diversos recursos presentes na ferramenta, embora alguns não sejam necessários para muitos projetos, é importante o usuário conhecer a capacidade do supervisório.

- Alertas

Uma característica bastante útil do ScadaBR é a função de alertas, onde busca indicar por meio de bandeiras coloridas e efeitos sonoros a ocorrência de notificações sobre a execução do sistema. Sempre que necessário um ícone de uma bandeira ficará piscando e será associada uma descrição próximo ao centro do cabeçalho. A cor da bandeira indicará a severidade do alerta, sendo Azul para notificações, Amarela para Alertas Urgentes, Laranja para Alertas críticos e Vermelho para possível risco de vida.

Este função é importante para manter o operador sempre informado do estado do sistema, permitindo tomar alguma ação principalmente em casos mais graves.

Existe também a possibilidade de personalizar os alertas, atribuindo sons diferentes para cada tipo de situação.

- Tratadores de Eventos

Um tratador de eventos é um mecanismo que permite a detecção de eventos no sistema e a partir de uma configuração do usuário responder ao evento automaticamente. Existem tanto eventos definidos pelo sistema como definidos pelo usuário. Eventos definidos pelo sistema incluem erros de operação de datasources, logins de usuários, e inicialização e parada do sistema. Eventos definidos pelo usuário incluem detectores de valor ou estado, eventos agendados e eventos auditados, que ocorrem quando usuários fazem alterações nas configurações da aplicação.

Uma vez que um evento foi detectado, é manipulado por tratadores, que definiram a ação a ser executada, como por exemplo, enviar um e-mail com informações do estado atual ou acionar um atuador quando um sensor atingir um determinador estado ou valor.

Figura 7 - Tratadores de Eventos

Fonte: Print Screen do Software ScadaBR

- Representação Gráfica

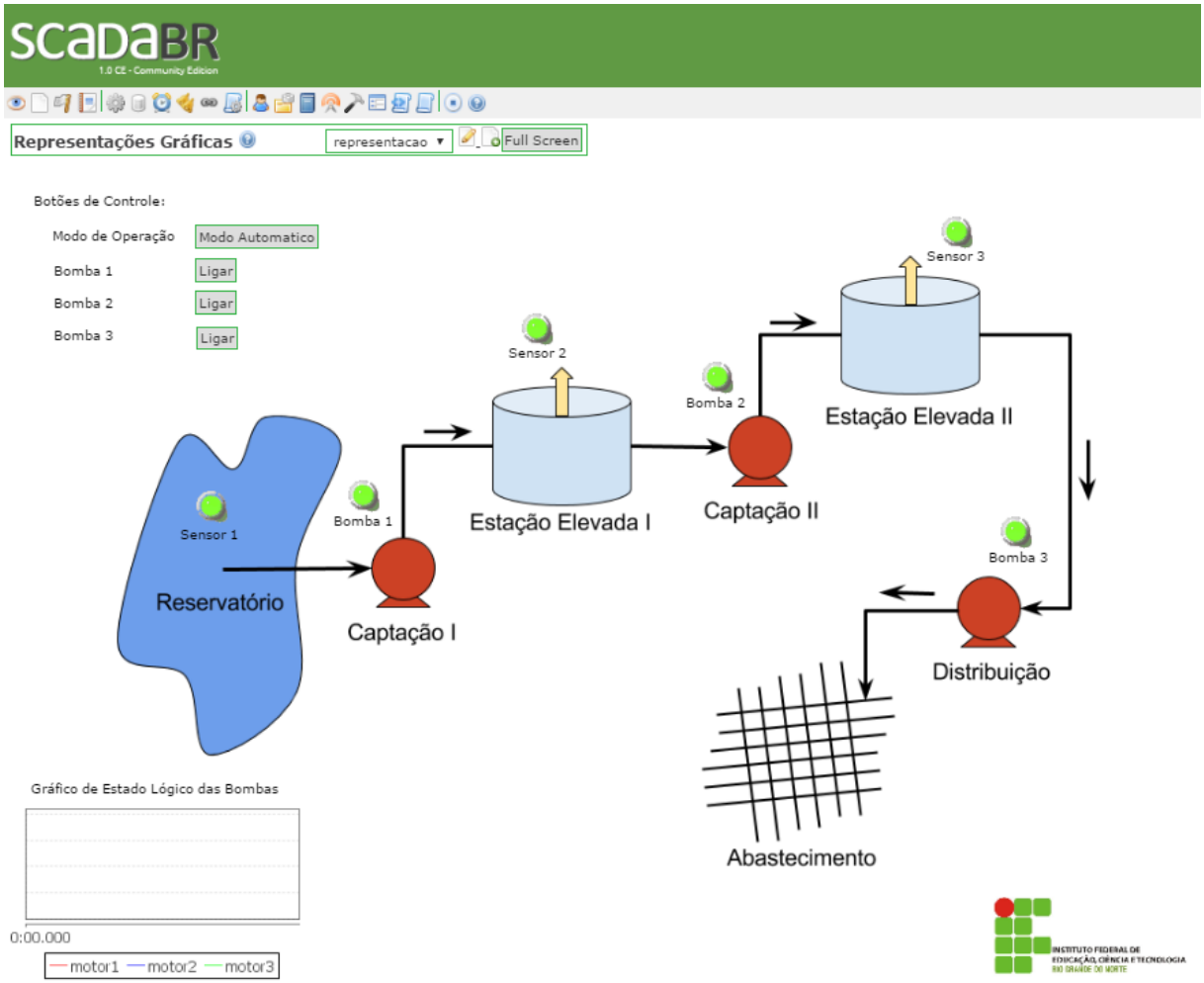
Sem dúvida uma das funções mais usadas no ScadaBR, é a geração de componentes gráficos para representar a aplicação, tornando o controle e monitoramento do processo mais interativo.

O usuário pode inserir diversos tipos de componentes sejam botões, gráficos ou imagens. Em alguns casos e preferencialmente recomendado, é possível até

simular o processo como seria na realidade, criando componentes que interagem dinamicamente ao longo do tempo.

Sempre que um componente é adicionado a representação, temos que o relacionar com um datapoint. É através disso, que os sensores e atuadores podem trocar dados com a representação.

Figura 8 - Representação Gráfica



Fonte: Print Screen do Software ScadaBR

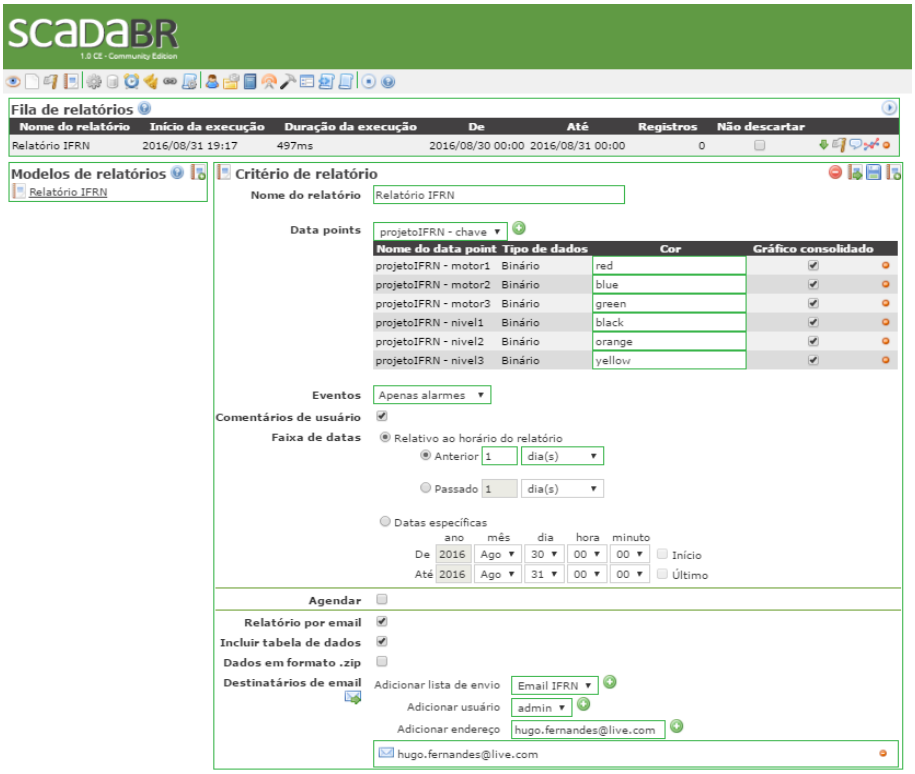
- Relatórios

O ScadaBR possui um gerador de relatórios próprio, onde se pode selecionar datapoints para serem incluídos no relatório mostrando o histórico dos valores e estados associados.

É possível, agendar relatórios para serem gerados automaticamente, como também incluir eventos, para que documentos sejam criados quando uma determinada ocorrência acontecer ou quando dados esperados sejam coletados após um tempo de execução.

Outra funcionalidade bem interessante, está no fato dos relatórios poderem ser enviados por e-mail utilizando uma lista de envio de e-mails para a qual o sistema enviará instâncias do relatório gerado. O conteúdo desses e-mails podem ser formatados de diversas maneiras, como também, exportado em formato CSV, que é bastante utilizado na criação de Bases de Dados.

Figura 9 - Relatórios



Fonte: Print Screen do Software ScadaBR

- Scripts

Para criar Scrips no ScadaBR, deve-se utilizar um componente chamado de Meta Datasource. Ele tem esse nome por sua capacidade de combinar pontos existentes em novos. Ao invés de obter sua informação de uma fonte externa, utiliza valores de outros pontos, permitindo manipular ou formatar os dados de um ponto de maneiras arbitrárias pelo usuário.

Antes de criar os Scripts é necessário ter inserido primeiro os datapoints que serviram de referencia para as rotinas programadas. Os Scripts são escritos em linguagem de programação C e podem retornar valores dos mais diversos tipos.

No código 2, é mostrado um exemplo de script que pode ser desenvolvimento para retornar resultados pré-determinados a partir de dados de humidade (humi_var) e temperatura (temp_var).

Código 2 - Exemplo de Script

```
1  a = "Clima Quente e Húmido";
2  b = "Clima Quente e Seco";
3  c = "Clima Frio e Húmido";
4  d = "Clima Frio e Seco";
5  e = "Clima Atual";
6  if (temp_var.value > 25 && humi_var.value > 80 )
7          e = a;
8  if (temp_var.value > 25 && humi_var.value < 80 )
9          e = b;
10 if (temp_var.value < 25 && humi_var.value > 80 )
11         e = c;
12 if (temp_var.value < 25 && humi_var.value < 80 )
13         e = d;
```

```
14  return e;
```

Fonte: Elaborado pelos autores, (2016).

Assim, dependendo do valor armazenado nos Data Points o Script retorna a mensagem implementada. No código 2, os valores retornados são do tipo texto, mais também podem ser de outros tipos, como numéricos ou digitais.

- Banco de Dados

Um recurso que atualmente está ganhando bastante popularidade, principalmente em aplicações que necessitam de capacidade em analise de dados, é o suporta a sistemas gerenciadores de banco de dados.

Até o momento o ScadaBR permite a integração com dois sistemas de gerenciamento de banco de dados, sendo eles o MySQL e o Derby. Esse ultimo já vem configurado por padrão, principalmente pela facilidade de não precisar inserir configurações adicionais.

3. PROCEDIMENTOS METODOLÓGICOS

3.1. Caracterização do Objeto de Estudo

Nesta parte, relaciona-se o objeto de estudo com o mercado, destacando os principais elementos da atividade, as suas particularidades, seus ramo de atuação e seus principais desafios.

Dessa forma, podemos caracterizar o objeto de estudo nos seguintes tópicos:

a) Forma de atuação: Por se tratar de um trabalho académico, o Capital utilizado para elaboração desse estudo foi exclusivamente próprio dos membros;
b) Data de instituição da organização: O estudo se iniciou alguns meses depois do termino das aulas na academia (IFRN), onde inicialmente o trabalho utilizaria CLPs (Controlador Lógico Programável) no controle dos processos. No entanto, depois de algumas discussões, foi decidido a substituição pela plataforma Arduino, tendo em vista que, os membros já possuíam experiência com a mesma e o projeto tornara-se mais acessível financeiramente. Desde então, os estudos seguiram essa linha de pesquisa;
c) Natureza Atual do Objeto: A natureza atual do estudo é a pesquisa e implementação das tecnologias;
d) Produtos e Processos envolvidos: Na elaboração do estudo foram adquiridos diversos produtos que auxiliaram, ou em alguns casos, tornaram possível o desenvolvimento do projeto. Os principais processos realizados foram de pesquisa e aplicação das tecnologias;
e) Força de trabalho: O principal capital humano que contribuiu nas diversas atividades realizadas, foram dos membros e orientador;
f) Clientes, mercados e concorrência: A priori um dos principais clientes, como também concorrentes que teriam interesse nesse estudo seria a CAERN (Companhia de Águas e Esgotos do RN), tendo em vista que, o projeto surgiu de um problema enfrentado por essa empresa.

g) Fornecedores e insumos: A maior parte dos produtos e equipamentos adquiridos para a implementação desse estudo, foram encontrados na própria região do Seridó (Interior do RN).

3.2. Método da Pesquisa

Nesse ponto, iremos classificar a pesquisa considerando uma serie de critérios. A seguir, apresentam-se alguns tipos de classificação:

- Quanto à natureza podem ser considerada aplicada, por se tratar da utilização de tecnologias no desenvolvimento de um projeto;
- Quanto a forma de tratamento e interpretação dos dados, podem ser considerada qualitativa, visto que, as analises não são realizadas baseando-se em dados estatísticos;
- Do ponto de vista dos objetivos propostos pode-se classificar como descritiva, pois tem por finalidade observar, registrar e analisar os fenômenos;
- Em relação à técnica de coleta dos dados pode ser considerada pesquisa-ação, pois se trata de um tipo de pesquisa que tenta unir a pesquisa à prática, isto é, desenvolver o conhecimento e a compreensão como parte da prática.

Capitulo 4: Projeto

1. MATERIAIS UTILIZADOS

- Arduino UNO

Figura 10 - Arduíno UNO R3

Fonte: https://pt.wikipedia.org/wiki/Arduino

- Atuadores: Bombas

O equipamento de atuação do sistema é uma eletrobomba para pequenas aplicações, onde seu nome de mercado é RS-360SH. A minibomba RS-360SH é ideal para pequenas aplicações e não deve ter regime de operação muito demorado. Possui a capacidade de trabalhar sem está mergulhada no liquido, sendo assim uma bomba de sucção à vácuo.

Figura 11 - Minibomba RS-360SH

Fonte: http://www.banggood.com/pt/Mini-Micro-DC-9V-RS-360SH-Water-Priming-Pump-p-87577.html

O projeto busca a elevação do nível da água para uma estação elevatória com o objetivo de utilizar o sistema com a máxima eficiência possível, dessa forma evitando a sobrecarga dos atuadores. Dessa maneira, cada eletrobomba será acionada individualmente e em momentos diferentes, evitando o acionamento simultâneo dos atuadores.

- Modulo Relé

Este Módulo Relé 5V com 4 canais permite integração com uma ampla gama de microcontroladores. A partir das saídas digitais pode-se, através do relé, controlar cargas maiores de dispositivos como motores.

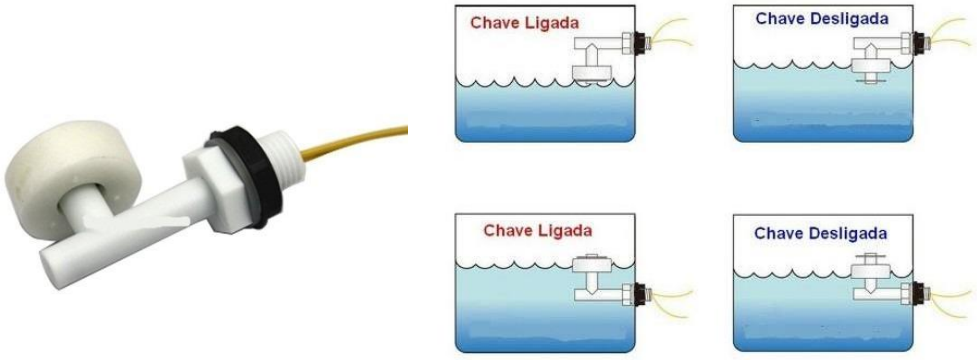
Figura 12 - Módulo Relé

Fonte: http://www.filipeflop.com/pd-c0ce5-modulo-rele-5v-4-canais.html

- Sensores de Nível

Os sensores são outra parte de essencial importância no projeto. Eles são os componentes capazes de verificar o nível da água nos recipientes.

Figura 13 - Sensor de Nível



Fonte: http://produto.mercadolivre.com.br/MLB-694413729-sensor-nivel-lateral-agua-arduino-microcontrolador-aquario-_JM

Os sensores funcionam como uma chave, atuando vezes como chave fechada, vezes com chave aberta, dessa maneira abrindo e fechando a passagem da corrente elétrica.

No estado ligado, o sensor permite a passagem a corrente, neste momento a boia de verificação de nível deve encontra-se em final de curso. No estado fechado, o sensor faz o processo contrario, cortando a passagem da corrente, a boia encontra-se no inicio do curso.

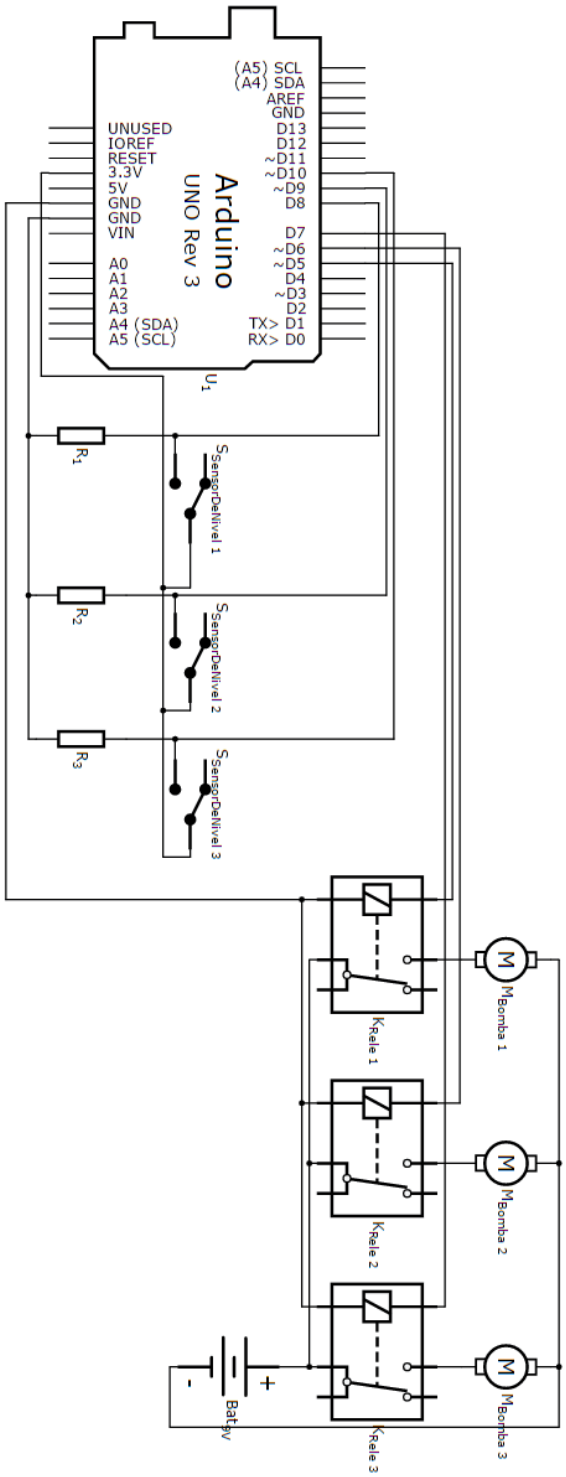
Um resistor foi utilizado para reduzir a corrente enviada para a placa arduíno, isso foi necessário para evitar danos ao equipamento. O tensão aplicada no resistor é de 3,3 volts, para isso buscamos uma resistência de 330 ohms resultando em um corrente de 10 mA.

2. SOFTWARES UTILIZADOS

- Windows 10
- Arduino IDE 1.6.9
- ScadaBR 1.0
- JDK 6

3. ESQUEMA ELETRICO DO PROJETO

Figura 14 - Esquema elétrico



Fonte: Elaborado pelos autores, (2016).

4. INSTALAÇÃO DA APLICAÇÃO

Instale o Arduino IDE versão 1.6.9, especificando o local de instalação. Essa versão da IDE do Arduino, não apresentou nenhum problema na utilização da biblioteca do ModBus, entretanto, versões anteriores podem não ser ideais ao projeto.

Instale o JDK 6, para a utilização do ScadaBR. Mesmo existindo outras versões mais atuais do kit de desenvolvimento do Java, é recomendado pela equipe do ScadaBR a instalação dessa versão, pois muitas funcionalidades ainda não foram atualizadas pela equipe do Projeto ScadaBR.

Instale o ScadaBR na máquina especificando o local de instalação. O projeto do ScadaBR é uma aplicação Web, ou seja, executada no Browser (Navegador), sendo necessária a instalação de um servidor web na máquina. Por padrão, ao instalar o ScadaBR o Tomcat6 é instalado junto na porta 8080. Vale lembrar aqui, que se na máquina conter outra aplicação executando na mesma porta, vai ocorrer conflito e ambos os serviços serão prejudicados. Assim, busque evitar isso.

Outro ponto que vale ser citado, é que no Windows nem sempre ao executar o ScadaBR o servidor web será executado junto. Dessa maneira, se isso ocorrer procure executar manualmente o servidor web, que por padrão está contido no diretório onde foi instalado o ScadaBR (/ScadaBR/bin/tomcat6.exe). Lembrando-se de executa-lo como administrador.

Por fim, a aplicação funcionou sem muitos problemas no Windows 10. No inicio, foi estudada a proposta de se desenvolver a aplicação em um ambiente GNU/Linux, no entanto, foi visto em fóruns e em tutoriais de comunidades que estudam Arduino e ScadaBR, que essa proposta não seria ideal, tendo em vista que, um ambiente GNU/Linux não possuía componentes para a execução do projeto. Assim, foi adotado o Windows como ambiente de desenvolvimento.

5. DESENVOLVIMENTO DO CÓDIGO ARDUINO

O principal componente dessa seção é a biblioteca do ModBus, que pode ser encontrada na pasta do projeto no github (https://github.com/hugaofernandes/arduino_modbus.git), com o nome de 'modbus.h', ou na internet sem muita dificuldade. Essa biblioteca contém todas as funções necessárias para a comunicação entre o Arduino e o ScadaBR. O outro arquivo necessário, está especificado com o nome 'sketch_jun22a.ino', este pode ser definido como o arquivo principal que chama a biblioteca do modbus e executa o código desenvolvido para a aplicação. Para deixar este trabalho mais legível, os códigos não serão exibidos na integra, entretanto o leitor pode a qualquer momento consultar os arquivos completos na pasta do projeto.

Código 3 - Chamada das bibliotecas necessárias para comunicação

```
1  #include "modbus.h"
2  void configure_mb_slave(long baud, char parity, char txenpin);
3  int update_mb_slave(unsigned char slave, int *regs, unsigned int regs_size);
4  enum {
5          COMM_BPS = 9600,
6          MB_SLAVE = 1,
7          PARITY = 'n'
8  };
9  enum {
10         modoOperacao,
11         bomba1Regs,
12         bomba2Regs,
13         bomba3Regs,
14         sensor1Regs,
15         sensor2Regs,
```

```
16          sensor3Regs,
17          MB_REGS
18  };
19  int regs[MB_REGS];
20  int ledPin13  =  13;
21  int bomba1 = 5;
22  int bomba2 = 6;
23  int bomba3 = 7;
24  int sensor1 = 8;
25  int sensor2 = 9;
26  int sensor3 = 10;
27  unsigned long millisAnterior = 0;
```

Fonte: Elaborado pelos autores, (2016).

Como pode ser visto no Código 3, a linha 1 mostra a importação da biblioteca ModBus. Logo em seguida, nas linhas 2 e 3 vemos a chamada das funções de configuração ModBus. Das linhas 4 à 27 basicamente ocorre são definições de variáveis, onde se pode destacar as variáveis COMM_BPS e MB_SLAVE que definem a taxa de transmissão da conexão e endereço do escravo na rede ModBus, respectivamente. Essas informações serão necessárias para configurar o ScadaBR futuramente. Outro ponto importante nesse código é o conteúdo entre as linhas 9 à 18, que se trata dos registradores que armazenaram os comandos de botões e estados dos Atuadores e Sensores. A ordem que eles são listados é vital, pois dessa maneira são definidos os valores de Offset através do endereço na enumeração, no caso variando de 0 na linha 10 até 6 na linha 16. As linhas 17 e 19 definem o número de registradores e os organizam em uma lista.

As variáveis das linhas 20 à 26 integram o código com o esquema elétrico, conectando os equipamentos nos pinos especificados. A linha 27, implementa junto

com a função “tempoDeExecução” um mecanismo de controle de falhas, não deixando uma bomba ficar ligada por mais de dois minutos, nesse caso.

Código 4 - Função Setup

```
29 void setup() {
30         configure_mb_slave(COMM_BPS, PARITY, 2);
31         pinMode(ledPin13,  OUTPUT);
32         pinMode(bomba1, OUTPUT);
33         pinMode(bomba2, OUTPUT);
34         pinMode(bomba3, OUTPUT);
35         pinMode(sensor1, INPUT);
36         pinMode(sensor2, INPUT);
37         pinMode(sensor3, INPUT);
38 }
```

Fonte: Elaborado pelos autores, (2016).

Como todo programa Arduino deve possuir uma função Setup, o seu principal proposito é especificar as configurações para execução do código. Em Código 4, vemos essa função, dando destaque a linha 30 onde é executada a função “configure_mb_slave”. Os parâmetros ditam a taxa de transmissão da conexão, a utilização de paridade e o pino do arduino que será responsável pela comunicação em uma rede RS485. As demais linhas, definem a função dos pinos na placa.

Código 5 - Função Loop

```
39 void loop() {
40         switch (regs[modoOperacao]) {
41                 case 1: { // Modo Automatico
42                         automatizar(sensor1Regs, sensor1, bomba3Regs, bomba3);
43                         automatizar(sensor2Regs, sensor2, bomba1Regs, bomba1);
```

```
44                          automatizar(sensor3Regs, sensor3, bomba2Regs, bomba2);
45                          break;
46                  }
47                  case 0: {  // Modo Manual
48                          manual();
49                          break;

50                  }
51          }
52  }
```

Fonte: Elaborado pelos autores, (2016).

A função Loop, busca executar o código no tempo. Ela funciona como um laço, sempre executando o que está dentro de seu escopo. O conteúdo em Código 5, trata-se de uma estrutura de decisão. Na linha 40, é mostrado um teste lógico que dependendo do valor assumido pelo registrador “modoOperacao”, o modo de execução da aplicação é definido. Se o resultado do registrador for verdadeiro, ou seja, assumir o valor 1, o sistema entra no modo automatizado, desenvolvendo o código das linhas 41 à 46. Caso contrario, as linhas de 47 à 50, fazendo o sistema à entrar no modo manual.

Código 6 - Função Automatizar

```
53  void automatizar(int sensorRegs, int sensor, int bombaRegs, int bomba) {
54          digitalWrite(ledPin13, regs[modoOperacao]);
55          regs[bomba1Regs] = desligarBomba(bomba1);
56          regs[bomba2Regs] = desligarBomba(bomba2);
57          regs[bomba3Regs] = desligarBomba(bomba3);
58          regs[sensorRegs] = verificacaoDeNivel(sensor);
59          update_mb_slave(MB_SLAVE, regs, MB_REGS);
60          millisAnterior = millis();
```

```
61          while (regs[modoOperacao] and !regs[sensorRegs] and
            tempoDeExecucao(millisAnterior)) {
62                  regs[bombaRegs] = ligarBomba(bomba);
63                  regs[sensorRegs] = verificacaoDeNivel(sensor);
64                  update_mb_slave(MB_SLAVE, regs, MB_REGS);
65          }
66          regs[bombaRegs] = desligarBomba(bomba);
67          update_mb_slave(MB_SLAVE, regs, MB_REGS);
68  }
```

Fonte: Elaborado pelos autores, (2016).

No código 6, começa desligando todas as bombas, está funcionalidade é importante para evitar o acionamento simultâneo dos atuadores, tendo em vista que, isso pode causar danos de sobrecarga a rede elétrica do sistema. Em seguida, é verificado o nível do reservatório e atualizado os valores de estado dos registradores pela função “update_mb_slave”. Essa função é essencial para a implementação do projeto, pois é através dela que os estados dos registradores, são informados ao supervisório ScadaBR usando a rede ModBus. Nas linhas 61 à 65, a bomba passada como parâmetro pela função é acionada enquanto o modo de operação continue automático, o nível do reservatório ainda esteja baixo e não tenha se passado dois minutos desdá execução dessa instrução (Essa medida é necessária para evitar que em casos de falhas, o sistema não consiga desligar uma bomba, causando grande desperdício de água). A execução entra nesse laço, mantendo a bomba acionada enquanto nenhuma das condições forem quebradas. Finalmente na linha 66, logo depois do laço ser encerado, a bomba é desligada.

Código 7 - Função Manual

```
69  void manual() {
70          digitalWrite(ledPin13, regs[modoOperacao]);
71          while (regs[bomba1Regs] and !regs[sensor2Regs] and !regs[modoOperacao]
            and tempoDeExecucao(millisAnterior)){
```

```
72                  regs[bomba1Regs] = ligarBomba(bomba1);
73                  regs[bomba2Regs] = desligarBomba(bomba2);
74                  regs[bomba3Regs] = desligarBomba(bomba3);
75                  regs[sensor2Regs] = verificacaoDeNivel(sensor2);
76                  update_mb_slave(MB_SLAVE, regs, MB_REGS);
77          }
78          regs[sensor2Regs] = verificacaoDeNivel(sensor2);
79          regs[bomba1Regs] = desligarBomba(bomba1);
80          update_mb_slave(MB_SLAVE, regs, MB_REGS);
81          while (regs[bomba2Regs] and !regs[sensor3Regs] and !regs[modoOperacao]
            and tempoDeExecucao(millisAnterior)){
82                  regs[bomba2Regs] = ligarBomba(bomba2);
83                  regs[bomba1Regs] = desligarBomba(bomba1);
84                  regs[bomba3Regs] = desligarBomba(bomba3);
85                  regs[sensor3Regs] = verificacaoDeNivel(sensor3);
86                  update_mb_slave(MB_SLAVE, regs, MB_REGS);
87          }
88          regs[sensor3Regs] = verificacaoDeNivel(sensor3);
89          regs[bomba2Regs] = desligarBomba(bomba2);
90          update_mb_slave(MB_SLAVE, regs, MB_REGS);
91          while (regs[bomba3Regs] and !regs[sensor1Regs] and !regs[modoOperacao]
            and tempoDeExecucao(millisAnterior)){
92                  regs[bomba3Regs] = ligarBomba(bomba3);
93                  regs[bomba1Regs] = desligarBomba(bomba1);
94                  regs[bomba2Regs] = desligarBomba(bomba2);
95                  regs[sensor1Regs] = verificacaoDeNivel(sensor1);
96                  update_mb_slave(MB_SLAVE, regs, MB_REGS);
97          }
98          regs[sensor1Regs] = verificacaoDeNivel(sensor1);
99          regs[bomba3Regs] = desligarBomba(bomba3);
100         update_mb_slave(MB_SLAVE, regs, MB_REGS);
```

```
101 }
```

Fonte: Elaborado pelos autores, (2016).

Em código 7, podemos ver a função manual, que permite ao operador do sistema acionar manualmente uma bomba via software por um botão no ScadaBR. A primeira vista está função pode parecer complexa, entretanto se trata apenas de uma cadeia de decisões, verificando que botão no ScadaBR está pressionado e acionando a bomba respectiva a ele. Os blocos de decisão podem ser encontrados nas linhas 71 à 77, 81 à 87 e 91 à 97. O funcionamento é similar, enquanto o botão no ScadaBR estiver ligado, o nível do sensor que corresponde a bomba permanecer em estado baixo e o tempo de execução do laço não ultrapassar 2 minutos, a bomba em questão será acionada e as demais desligadas para evitar sobrecarga. Por fim, cada bomba é desligada ao final da função.

Código 8 - Funções de Acionamento de Bombas

```
102 int ligarBomba(int bomba) {
103         digitalWrite(bomba, LOW);
104         return 1;
105 }
106
107 int desligarBomba(int bomba) {
108         digitalWrite(bomba, HIGH);
109         return 0;
110 }
```

Fonte: Elaborado pelos autores, (2016).

Em código 8, são mostradas as funções de acionamento dos atuadores. Nelas são passadas como parâmetro o pino do relé que aciona a bomba correspondente, logo em seguida a função “digitalWrite” atribui o estado lógico

necessário para ligar ou desligar a bomba. Finalmente, as funções retornam o valor 0 (desligado) ou 1 (ligado) que serão atribuídos aos registradores de estado no ScadaBR.

Código 9 - Função de Verificação de Nível

```
111 int verificacaoDeNivel(int sensor) {
112         int media = 0;
113         while (digitalRead(sensor)) {
114                 media++;
115                 if (media <= 3) {
116                         return 0;   // nivel baixo (Reservatorio Baixo)
117                 }
118         }
119         return 1;  // nivel alto (Reservatorio Cheio)
120 }
```

Fonte: Elaborado pelos autores, (2016).

No código 9, é apresentada a função que realiza a medição do nível de água nos reservatórios pelos sensores. A função “digitalRead” efetua a leitura do sensor especificado pelo parâmetro, que se trata do pino onde o sensor está instalado, retornando o valor 0 (Nível Baixo) ou 1 (Nível Alto). Para evitar possíveis falhas na leitura do nível pelo sensor, foi implementada uma média, no qual o resultado só retornará positivo depois de 3 verificações de nível alto.

Código 10 - Função Tempo de Execução

```
121 int tempoDeExecucao(unsigned long tempo) {
122         unsigned long mili = millis();
123         if (mili >= tempo) {
124                 if ((mili - tempo) < 120000) {
```

```
125                             return 1; }
126                     return 0; }
127             else {
128                     if ((tempo - mili) < 120000) {
129                             return 1; }
130                     return 0; }
131 }
```

Fonte: Elaborado pelos autores, (2016).

No código 10, vemos a função que mede o tempo da execução dos atuadores. Está função inicia recebendo como parâmetro o valor em milissegundos do tempo de execução do programa desde sua execução, está tarefa é realizada pela função “Millis”. Na linha 122, essa função é executada novamente para capturar o tempo de execução do programa. Nas demais linhas, é implementada uma subtração entre o tempo medido atualmente e o tempo medido antes da execução da função de acionamento da bomba. Está atividade é executada por várias vezes, verificando o tempo que um atuador fica acionado, se esse tempo for superior a dois minutos a função retorna o valor 0, parando a bomba.

Finalmente, o código pode ser enviado para a placa Arduino. No projeto em questão, foi utilizado a placa Arduino UNO pelo motivo de ser um componente barato e simples de programar, entretanto, pode ser utilizado em outras plataformas Arduino sem muitas alterações.

6. CONFIGURAÇÕES DO SCADABR

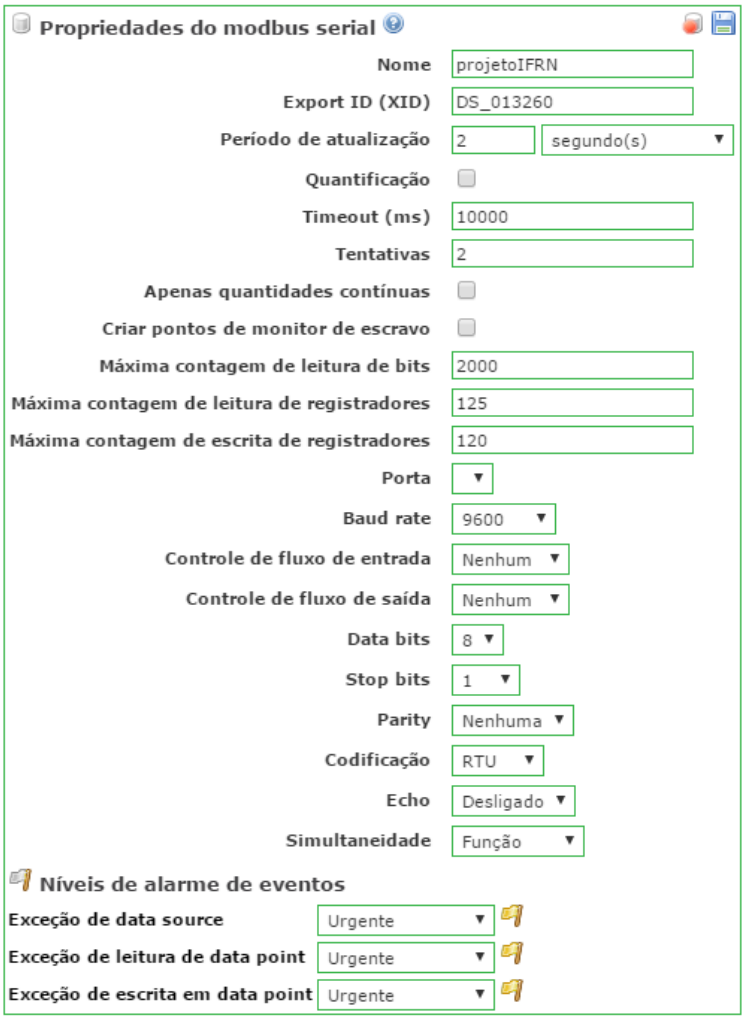
Logo depois da instalação do ScadaBR, ao executa-lo a primeiro tela a ser mostrada é um pequeno manual listando as especificações do software. Assim, a primeira ação de fato é efetuar o login na aplicação. Por padrão, o usuário (User Id) deve ser 'admin' e senha (Password) 'admin'.

Em seguida, o sistema já encontra-se pronto para configuração. O usuário deve criar um Data Source especificando o tipo de comunicação que o mesmo utilizará, no caso, a ModBus Serial.

Ao iniciar a configuração do Data Source, serão listados vários parâmetros onde se pode alterar dependendo da aplicação. Os mais importantes para nosso projeto, são o nome do Data Source, o Período de Atualização e a Porta de Comunicação. Para se definir a Porta de Comunicação, a placa Arduino deve está conectada a máquina, onde se marca a opção correspondente a porta utilizada. O Período de Atualização preferencialmente deve ser configurado para dois segundos, assim o sistema verificará o estado dos registradores a cada dois segundos. Dependendo da aplicação e da qualidade dos equipamentos esse período pode ser alterado.

Os outros parâmetros como, Baud Rate (Taxa de Transmissão), Data Bits (Tamanho do Pacote), Parity (Paridade), Codificação, podem ser definidos por padrão, ou seja, podem ser mantidos como estão. Essas configurações são importantes pois no código que está no Arduino são inseridos esses parâmetros, dessa forma, vale conferir se os valores listados aqui, estão em conformidade com os escritos no código fonte.

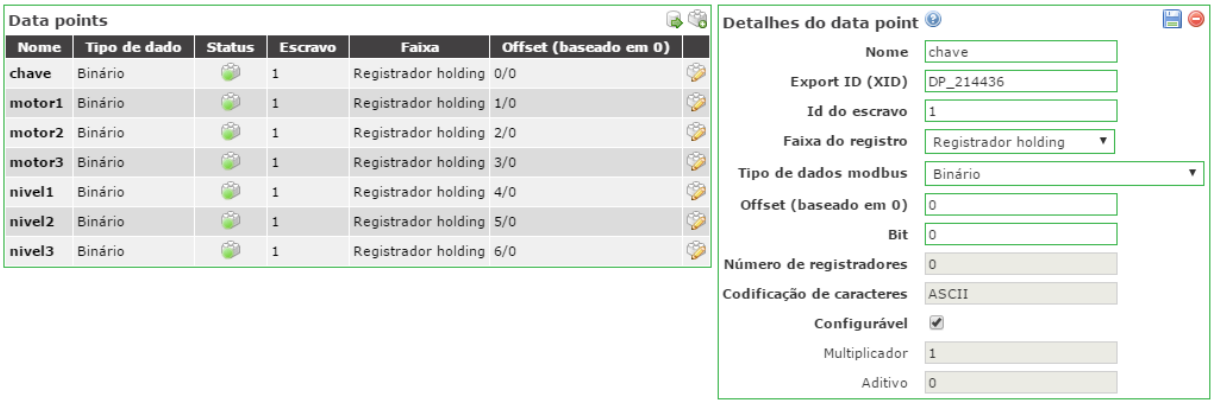
Figura 15 - Configuração do Data Source



Fonte: Print Screen do Software ScadaBR

O passo seguinte, é a criação dos Data Points. Os data points são os registradores que armazenam as informações dos sensores, botões e bombas do projeto. Eles são necessários para as funções de controle e supervisão do sistema. Ao configura-los, deve-se atentar para os parâmetros nome, tipo de dado e offset. No projeto em questão, os tipos de dados configurados foram como digitais. Os offsets merecem uma atenção especial, pois devem está relacionados com a variável definida no código que vai no Arduino, no caso do projeto, de 0 a 6.

Figura 16 - Configuração dos Data Points

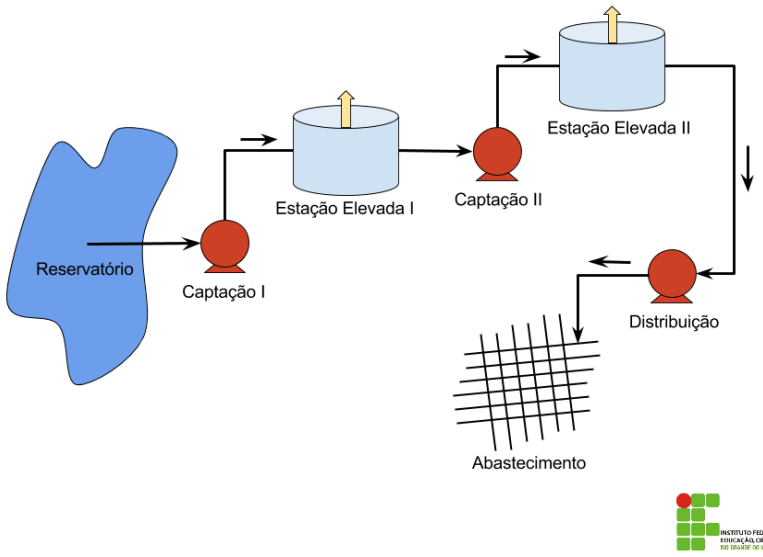


Fonte: Print Screen do Software ScadaBR

Terminada as configurações do data source e data points, o passo seguinte é a configuração das representações gráficas desses componentes. Essa tarefa é importante pois a partir dessa representação, é que o usuário vai poder controlar e supervisionar o estado da aplicação.

O primeiro passo para se representar a aplicação é escolher uma imagem que serva como plano de fundo do projeto. Ela não é necessária, mais pode ajudar na compreensão do que se quer desenvolver. Na figura 18, podemos ver o plano de fundo usado no projeto desse trabalho.

Figura 17 - Plano de Fundo

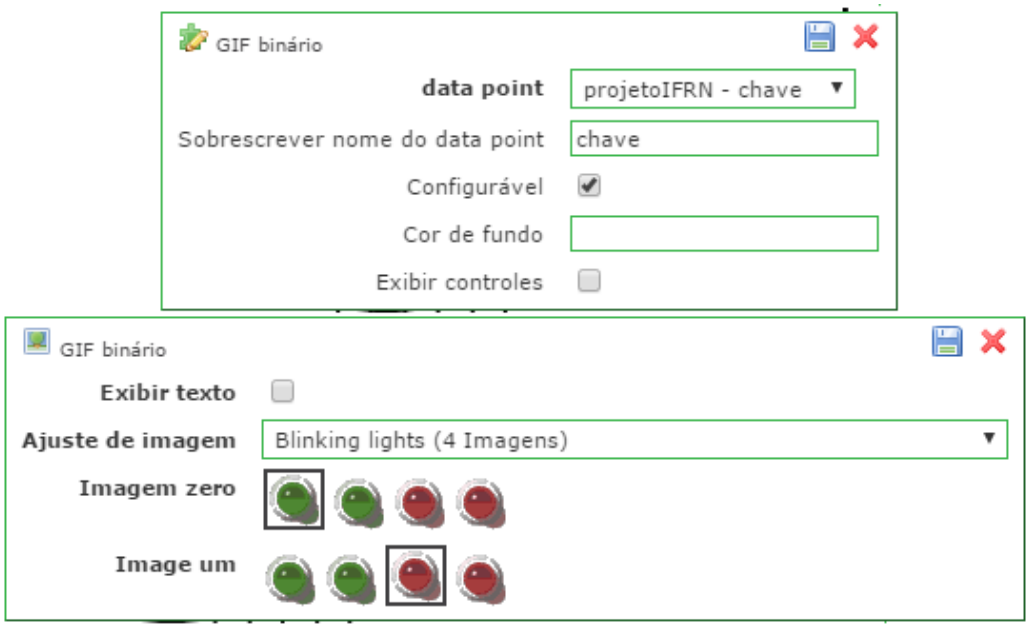


Fonte: Elaborado pelos autores, (2016).

Ainda em Representação Gráfica, temos que adicionar componentes para supervisão do sistema, como também mecanismos para controle da aplicação. O componente usado para visualizar a execução do projeto, foi o chamado “GIF Binário”, que mostra por meio de imagens (no caso, LEDs verdes e vermelhos) o estado do componente.

Ao adicionar um “GIF Binário”, devemos configurar dois parâmetros, primeiramente associar um Data Point ao componente e escolher imagens para servir como referência de mudança de estado, especificando os comportamentos para os valores 0 e 1. Na figura 19, podemos ver essas configurações.

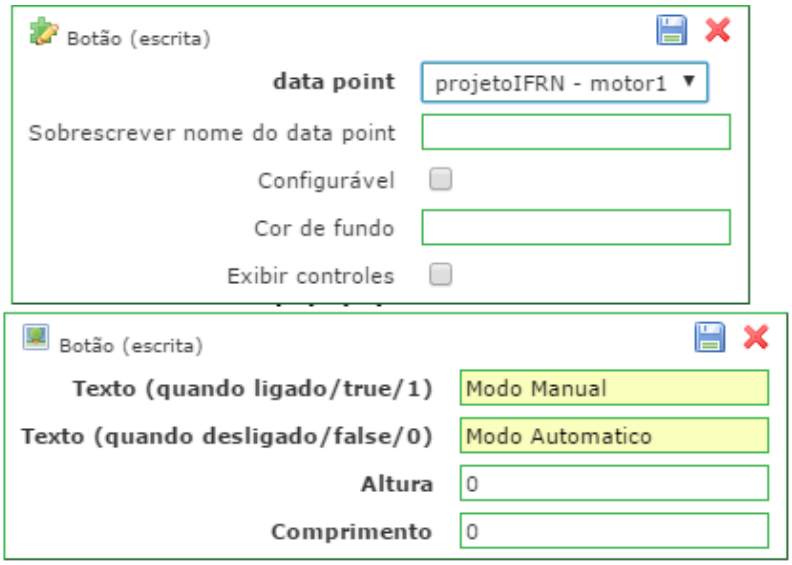
Figura 18 - Configurações do GIF Binário



Fonte: Print Screen do Software ScadaBR

Adicionando o componente “Botão Escrita”, podemos implementar botões que nos permitirá controlar o sistema durante sua execução. De maneira semelhante ao GIF Binário, o Botão Escrita necessita de duas configurações, a primeira é associar o componente a um Data Point correspondente e depois atribuir nomes aos estados lógicos 0 e 1. Na figura 20, podemos ver essas configurações.

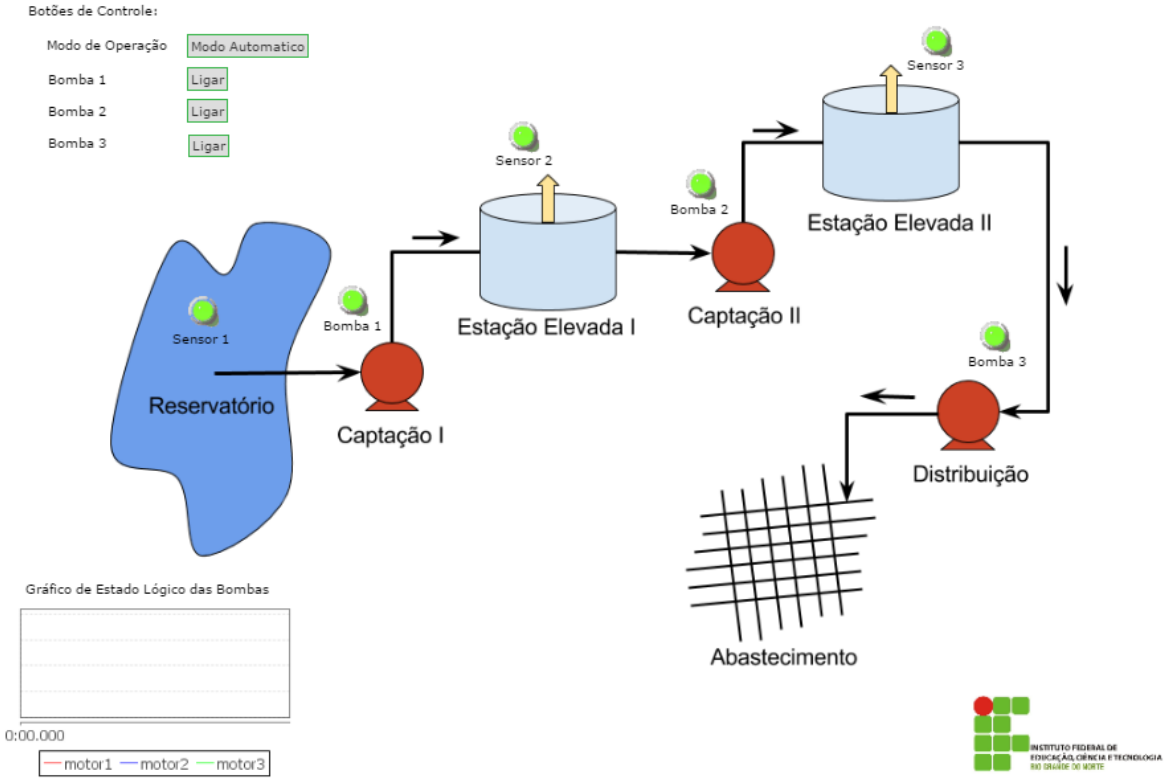
Figura 19 - Configurações do Botão Escrita



Fonte: Print Screen do Software ScadaBR

Após a criação de todos os componentes e associar aos Data Points correspondentes, só salvar as alterações. Na figura 21, podemos ver o resultado final da Representação Gráfica.

Figura 20 - Representação Gráfica

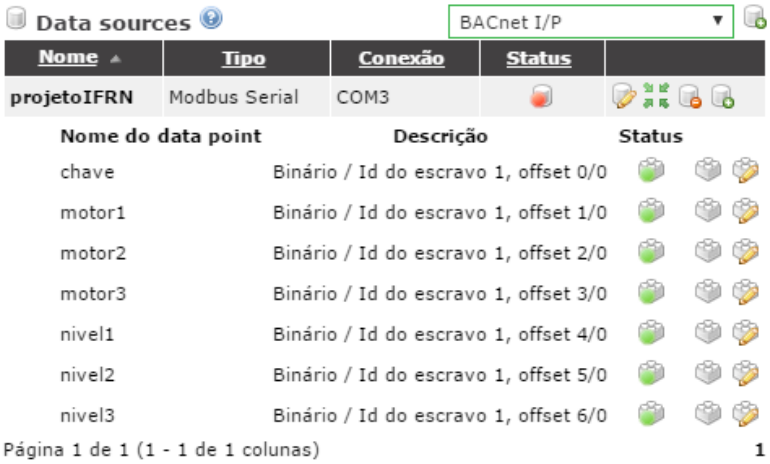


Fonte: Print Screen do Software ScadaBR

7. FUNCIONAMENTO DO SISTEMA

Ao terminar as configurações, o sistema estará pronto para uso. Na seção Datasources o usuário deve Habilitar a comunicação do sistema na opção status. Está ação inicia a comunicação entre o dispositivo escravo e o mestre da rede. Vale lembrar que os dispositivos devem está devidamente conectados ao computador, observe o campo conexão e verifique a porta que o equipamento está utilizando.

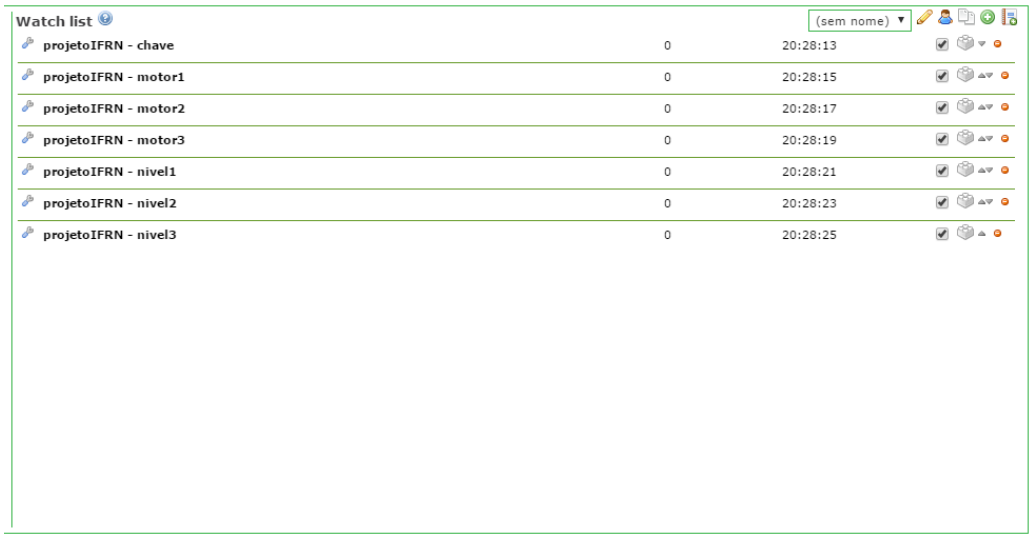
Figura 21 - Datasource



Fonte: Print Screen do Software ScadaBR

Em Seguida, o usuário deve verificar na seção Watch List se os Datapoints estão habilitados e atualizados. Esta verificação é necessária sempre que o sistema for reiniciado.

Figura 22 - Watch List



Fonte: Print Screen do Software ScadaBR

Por fim, se tudo foi realizado corretamente, o sistema estará em execução. O estado inicial da aplicação desenvolvida neste trabalho é a execução do modo manual. Os LEDs mostraram o estado dos sensores e atuadores, ficando vermelhos toda vez que estiverem ativos. Os botões à esquerda podem ser acionados para manipulação do projeto, sendo o botão chave para mudar o modo de operação da aplicação (Manual ou Automático), e os demais botões para acionar as bombas.

Uma vez estabelecida a comunicação entre o ScadaBR e o Arduino, o protocolo ModBus implementa o ScadaBR como o mestre da rede e a(s) placa(s) Arduino como escravo, desenvolvendo as tarefas enviadas pelo mestre.

4. CONSIDERAÇÕES FINAIS

Por fim, o projeto funcionou como esperado, realizando o objetivo de transportar água para uma estação elevada de forma autônoma. O protocolo ModBus junto com a plataforma Arduíno e o software ScadaBR se mostraram como excelentes ferramentas para o sucesso desse trabalho.

Entretanto, no decorrer das atividades foram enfrentados diversas dificuldades, como por exemplo, o fato de no inicio do estudo buscarmos desenvolver o projeto prático utilizando CLPs, o que acarretou no atraso da proposta, visto que, o acesso aos equipamentos para se trabalhar numa proposta desse tipo, seriam um impedimento tanto pelo custo, quanto pela disponibilidade.

Outro problema enfrentado, foi a falta de experiência na área de automação. Trabalhando inúmeras vezes utilizando o método da tentativa e erro na aplicação dos conceitos.

Um ponto que vale a pena ressaltar, é o custo necessário para o desenvolvimento de um protótipo didático. Os componentes utilizados foram adquiridos entre 2015 e 2016 onde parte foi comprada na própria cidade de Caicó/RN e parte na Internet. O valor total da aplicação girou em torno dos R$430,00. Assim, levando em consideração a escalabilidade do projeto, onde os equipamentos possam ser aplicados a um problema maior, poucos componentes necessitariam serem redimensionados, o que torna o custo na implementação notoriamente baixo para uma aplicação didática. Na tabela 14, podemos ver o valor de cada equipamentos usado no projeto.

Tabela 14 - Custos do Projeto

Equipamento	Quantidade	Valor
Plataforma de Madeira	1	R$ 25,00
Equipo	3	R$ 2,50
Condutores	5 Metros	R$ 1,00
Caixa de Passagem	2	R$ 5,00
Lacre (Engasga Gato)	6	R$ 0,50
Fita Isolante	1	R$ 3,00
Fonte 9 Volts	1	R$ 25,00
Bombas	3	R$ 51,45
Sensores de Nível	3	R$ 28,90
Relé	1	R$ 25,00
Arduino Uno R3	1	R$ 74,89

Fonte: Elaborado pelos autores, (2016).

Dessa forma, ficamos satisfeitos com os resultados obtidos e propormos como trabalhos futuros, a execução do projeto utilizando todos os recursos do software ScadaBR, como por exemplo, a função de envio de relatórios do sistema por e-mail e implantação de um banco de dados para armazenamento de status do sistema.

REFERÊNCIAS

MONK, Simon. **Trinta projetos com Arduino**. 2. ed. Porto Alegre. Bookman, 2014.

SOUZA, V, A. **O Protocolo ModBus.** Disponível em: <http://www.cerne-tec.com.br/Modbus.pdf>. Acessado em: 29/06/2015.

FREITAS, C,M. **Protocolo ModBus: Fundamentos e Aplicações.** Disponível em: <http://www.embarcados.com.br/protocolo-modbus>. Acessado em: 30/06/2015.

CERTI - Arduino com ScadaBR. Disponível em: <https://sites.google.com/a/certi.org.br/certi_scadabr/home/minicursos/arduino>. Acessado em: 30/06/2015.

ScadaBR - **Forums.** Disponível em: <http://www.scadabr.org.br/?q=forum>. Acessado em: 15/08/2015.

ScadaBR - **Manual do Software.** Disponível em: <http://www.scadabr.org.br>. Acessado em: 20/08/2015.

Simply Modbus - **Modbus ASCII vs Modbus RTU**. Disponível em: <http://www.simplymodbus.ca/ASCII.htm>. Acessado em: 01/08/2016.

MADEIRA, Daniel. **Estabelecendo Comunicação via protocolo ModBus com ScadaBR**. Disponível em: <http://www.embarcados.com.br/arduino-e-scadabr>. Acessado em: 01/08/2016.