Universidade Federal do Rio de Janeiro

Escola Politécnica

Departamento de Eletrônica e de Computação

Redes ZigBee e Programas Java Aplicados à Padronização e Melhoria da Qualidade dos Testes de Automação em Gasodutos.

Autor:

Vitor Sepulveda Boechat

Orientador:

Prof. Sergio Barbosa Villas-Boas, Ph. D.

Examinador:

Prof. Aloysio de Castro Pinto Pedroza, Dr.

Examinador:

Prof. Antônio Cláudio Gómez de Sousa, M. Sc.

DEL

Janeiro de 2010

DEDICATÓRIA

Dedico este trabalho, fruto de minha extrema motivação em me tornar um engenheiro, a todas as pessoas que me apoiaram direta ou indiretamente e que me deram forças, paz e alegria como meio indispensável para conclusão de minha graduação.

“Não precisamos saber pra onde vamos
nós só precisamos ir...”

Humberto Gessinger

AGRADECIMENTO

Agradeço, primeiramente, a Deus por ter iluminado meu caminho, me orientado e me dado forças durante toda minha jornada acadêmica.

Aos meus pais, Paulo Boechat e Maria Zelita Sepulveda, e minha irmã, Jéssica, pelo total e inquestionável apoio a minha vocação e pela sólida base humana, social e moral que me constitui.

A minha namorada, Zaila, pelo inconfundível zelo e apego a minhas metas e pelo amor, carinho e compreensão fundamentais na minha trajetória.

Ao meu irmão de coração, Eduardo Ribeiro, por todos momentos de alegria e pela constante motivação de meu desenvolvimento pessoal, profissional e acadêmico.

A toda minha família, em especial a minha tia Elisabeth Sepulveda, aos meus avós Manoel Sepulveda, Zelita Sepulveda, Marina Boechat e Euly Boechat (*in memoriam*), aos meus queridos primos Gustavo Lemos e Anderson Tavares por fortalecerem minha identidade e pelas alegrias.

Ao meu amigo Aluísio Telles, pelas palavras de apoio, incentivo pessoal e profissional; pela credibilidade e confiança em mim depositado, fundamentais em meu desenvolvimento.

A todos meus colegas de trabalho, em especial a Frederico Wegelin, Rafael Thadeo, Fabio Lima, Cleuber Queiroz e Paulo Tujal pela paciência, apoio e incentivo no desenvolvimento de minhas atividades, inclusive deste projeto. A Marcus Garcia e Amaury dos Santos pela compreensão e total apoio em minha formação profissional e acadêmica.

Aos meus amigos de faculdade Fernando Menegat, Rennan Roig, Marco Bril, André Cotrim, Rodrigo César e Danilo Cunha por contribuírem direta ou indiretamente com minha graduação.

A todas essas pessoas e as que aqui não foram citadas, meus sinceros votos de sucesso e meu: Muito obrigado.

RESUMO

O processo de Teste de Aceitação de Fábrica (TAF) constitui-se como parte fundamental no ciclo de vida de um determinado projeto. Com o objetivo de alinhar e solucionar algumas divergências contratuais entre o fornecedor e o cliente, na compra de um dado equipamento, este procedimento pode causar grande impacto no cronograma do empreendimento.

O sistema proposto por esse projeto final constitui-se em viabilizar uma execução do Teste de Aceitação de Fábrica em malhas de gasodutos de um modo automático reduzindo as chances de desconformidades, padronizando o procedimento de teste e aumentando a qualidade e conseqüentemente a segurança da instalação.

O automatismo do sistema foi alcançado utilizando um conjunto de tecnologias para poder simular desde a entrada física no campo até a sua respectiva variável supervisionada em um determinado centro de controle. Para simular a entrada física foi desenvolvido um sistema *wireless* baseado no protocolo ZigBee; Para simular um sistema supervisório e controlar as entradas físicas foi desenvolvido um programa em Java; Dessa forma é possível aplicar a um dado equipamento uma ampla gama de testes em um intervalo de tempo relativamente menor.

Palavras-Chave: ZigBee, ModBus, CLP, Java.

ABSTRACT

The process of Acceptance Test Factory (TAF) is as a key part in the life cycle of a certain project. In order to align and resolve some contractual disagreements between the supplier and customer, buying equipment, this can cause great impact on the development schedule.

The system proposed by this final project is to make the execution of the Acceptance Test Factory in gas pipelines in an automatic manner by reducing the chances of nonconformities in order to standardize the testing procedure and increasing the quality and therefore safety of the installation.

The control system was achieved using a set of technologies to simulate the physical input from the field to its respective variable in a given supervised control center. To simulate the physical input has developed a wireless system based on the ZigBee protocol, to simulate a system supervisory control and physical inputs has developed a Java program, this way you can apply to a specific equipment with a wide variety of tests on a range of time is relatively minor.

Key-words: ZigBee, ModBus, PLC, Java.

SIGLAS

AASD - Applied Agile Software Development

A/D – Conversor Analógico para Digital

AODV – Ad-hoc On-demand Distance Vector

API – Application Programming Interface

ASCII – American Code for Information Interchange

ATS – Automatic Tester System

D/A – Conversor Digital para Analógico

CLP – Controlador Lógico Programável

CPU – Unidade Central de Processamento

CRC – Cyclical Redundancy Check

CSMA/CD – Carrier Sense Multiple Access with Collision Detection

IEEE – Institute of Electrical and Electronics Engineers

I/O – Input/Output

ISM – Industrial, Scientific and Medical

LRC - Longitudinal Redundancy Check

LSB – Bit menos significativo

ModBus – Protocolo de comunicação desenvolvido pela Modicon

NA – Normalmente Aberta

NF – Normalmente Fechada

PAN – Personal Area Network

RF – Radio frequency

RTU – Remote Terminal Unit

SCADA – Supervisory Control and Data Acquisition

TAF – Teste de Aceitação de Fábrica

TAC – Teste de Aceitação de Campo

TCP/IP – Transmission Control Protocol/Internet Protocol

XBee – Módulo de comunicação com protocolo ZigBee embarcado

ZigBee – Protocolo de comunicação wireless desenvolvido pela ZigBee Alliance

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

1.1 – Tema

O tema do trabalho é o estudo e a criação de um sistema de testes de automação em malhas de gás natural. Neste sentido, pretende-se responder ao problema relacionado com a viabilidade em se criar um aplicativo computacional e uma interface de hardware capaz de automatizar o processo de testes. A hipótese inicial é que se o processo é passível de tratamento sistemático, então pode ser proposta uma arquitetura que o descreva e o conduza. Desta forma, é possível aumentar a qualidade e confiabilidade dos testes e também padronizar a forma como são conduzidos.

1.2 – Delimitação

Os objetos de estudo são os equipamentos que compõe a malha de gás e suas respectivas funções. O sistema está voltado à validação das lógicas e da comunicação dos equipamentos de campo e não da validação física e mecânica.

Como parâmetros do projeto foram utilizadas as lógicas de operação remota de válvulas, sendo perfeitamente válido a qualquer outro tipo de instrumentação de campo.

1.3 – Justificativa

O teste de aceitação de automação em malhas de gás usualmente é feito manualmente. A comunicação e o funcionamento das lógicas utilizadas são validados uma a uma comparando os resultados com o previsto no padrão de funcionamento do instrumento.

Neste sentido, o presente projeto é uma melhoria do processo de testes, buscando aumentar a qualidade, otimizar o tempo gasto, assim como padronizar o processo. Desta forma, a importância deste trabalho está relacionada ao aumento da confiabilidade do processo através de um modelamento dessa atividade.

1.4 – Objetivos

O objetivo geral é, então, propor um sistema capaz de automatizar o processo de validação das lógicas e comunicação dos instrumentos de campo. Desta forma, tem-se como objetivos específicos: (1) Modelar o procedimento de testes de um determinado equipamento – válvula – e implementá-lo em um aplicativo computacional; (2), Criar um interfaceamento wireless entre o aplicativo e o instrumento de campo ou seu controlador e; (3) Elaborar um "driver" para que o aplicativo leia o retorno do equipamento.

1.5 – Metodologia

O método é ágil, baseado em AASD (Applied Agile Software Development). Os requisitos serão atendidos progressivamente em versões sucessivas do programa. Uma primeira versão implementará a parte de comunicação via ModBus entre o CLP e o programa Java. Uma segunda versão implementará a comunicação entre o programa e a rede ZigBee. E finalmente um terceira versão modelará os testes utilizando-se da comunicação ModBus (para ler o retorno do CLP) e da rede ZigBee para simular o instrumento de campo. Será utilizado repositório de fontes subversion (SVM).

1.6 – Descrição

A organização e conteúdo dos capítulos visam estruturar e guiar o leitor para uma compreensão máxima do assunto abortado. Neste âmbito serão explicadas algumas partes de determinados assuntos com o intuito de enriquecer a argumentação e manter a leitura coesa e interessante.

No capítulo 2 será apresentada uma descrição geral da arquitetura do projeto, visando esclarecer e introduzir melhor o leitor ao princípio de funcionamento do sistema. Serão detalhadas as tecnologias utilizadas e suas respectivas justificativas para o objetivo proposto.

O capítulo 3 fará uma introdução e apresentará as principais características dos CLP's, abrangendo suas funcionalidades, comunicação, I/O e como o sistema se relaciona com o controlador.

No capítulo 4 será explicado o protocolo Modbus, suas funções e como o projeto se relaciona com esta tecnologia.

No capítulo 5 será introduzido o protocolo ZigBee e a descrição de seu funcionamento e suas principais características e seu papel na arquitetura do sistema. O capítulo 6 visa complementar este capítulo introduzindo os conceitos e características do módulo Xbee.

O capítulo 7 complementa os capítulos anteriores, visto que descreve todo o funcionamento do sistema com o leitor familiarizado com as tecnologias empregadas na execução do projeto. Inclui a relação da aplicação Java entre os diversos componentes citados nos capítulos anteriores e o modelamento de testes realizado.

No capítulo 8 serão apresentados os principais resultados e a conclusão.

Capítulo 2

Arquitetura do Sistema

2.1 – Introdução

O sistema baseia-se no conceito de simulação; Observando o equipamento a ser testado, CLP, como um bloco com I/O (Figura 2.1) é possível variar as entradas e monitorar as saídas de forma a se obter um modelo de funcionamento e se comparar com um padrão previamente estabelecido.

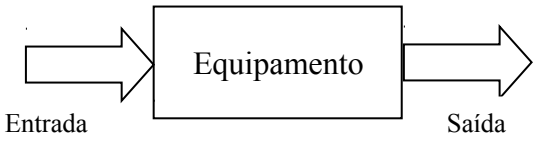


Figura 2.1 – Bloco com I/O

Normalmente estes equipamentos são feitos exclusivamente para determinadas aplicações, sendo assim não existe um teste único para todo o equipamento. O que ocorre é um conjunto de testes cada um com seus padrões e parâmetros de funcionamento diferentes. Este conjunto chama-se TAF (Teste de Aceitação de Fábrica).

Para atender ao proposto o sistema deste projeto, aqui definido ATS (Automatic Tester System), deve se conectar ao equipamento de modo a obter os dados da saída e gerar os dados na entrada. A figura 2.2 ilustra como a conexão é feita entre os sistemas.

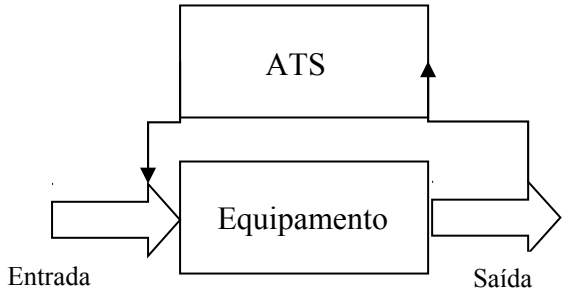


Figura 2.2 – Fluxograma geral

2.2 – Descrição

O projeto desde sistema se limita a testar os equipamentos de campo denominados CLP. Estes equipamentos são utilizados para realizar operações sobre uma determinada planta e/ou disponibilizar dados para uma determinada central de controle; No capítulo seguinte será apresentado de forma mais detalhada o conceito do CLP.

Para realizar esta tarefa o sistema comporta um conjunto de tecnologias capazes de simular toda a planta, desde o instrumento de campo até o sistema SCADA em que está sendo monitorado. O protocolo ZigBee constitui-se o protocolo principal entre o ATS as entradas do CLP; O protocolo ModBus sobre RS-232 é o utilizado pelo sistema para colher e enviar dados para o CLP. Ambos os protocolos serão detalhados nos capítulos subsequentes. A topologia básica do projeto é apresentada na figura 2.3, a seguir.

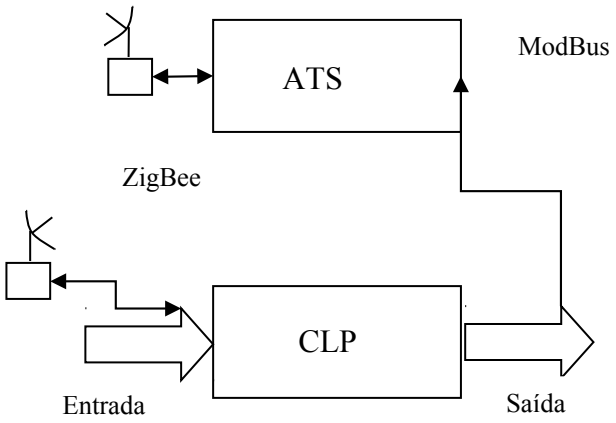


Figura 2.3 – Detalhamento das tecnologias

Capaz de manipular as entradas via protocolo ZigBee e ler as informações direto do CLP via protocolo modbus, o programa ATS desenvolvido em Java, consegue gerar inúmeras combinações de situações na entrada e comparar a saída com as especificações técnicas de funcionamento e os padrões em determinadas situações.

Para exemplificar um processo de testes realizados em um TAF, este projeto se limitou a fazer a validação das lógicas implementadas no CLP para monitoração e controle de válvulas. Desta forma as entradas do CLP simuladas via protocolo ZigBee desempenham o papel de chaves de fim de curso e a saída modbus é lida para controle da situação da válvula perante a simulação das entradas.

Em um sistema real as chaves de fim de curso das válvulas estariam diretamente ligadas ao CLP e este a um sistema de comunicação para aquisição de dados remotamente.

Capítulo 3

CLP

3.1 – Introdução

As empresas estão se reorganizando para atender as necessidades atuais de aumento de produtividade, flexibilidade e redução de custos. Destas metas surgiu à necessidade de os equipamentos se adequarem rapidamente às alterações de configurações necessárias para produzirem diversos modelos de produtos

Os Controladores Lógicos Programáveis, CLP's, são equipamentos eletrônicos utilizados em sistemas de automação flexível, sendo ferramentas de trabalho muito úteis e versáteis para aplicações em sistemas de acionamentos e controle, e por isso são utilizados em grande escala no mercado industrial. Permitem desenvolver e alterar facilmente a lógica para acionamento das saídas em função das entradas. Desta forma, podemos associar diversos sinais de entrada para controlar diversos atuadores ligados nos pontos de saída.

Podemos destacar as principais vantagens do CLP na utilização em projetos de automação: Praticidade, localização de falhas, flexibilidade, expansão e tempo de processamento.

3.2 – Funcionamento

Podemos apresentar a estrutura interna de um CLP dividida em três partes: entrada, processamento e saída (figura 3.1):

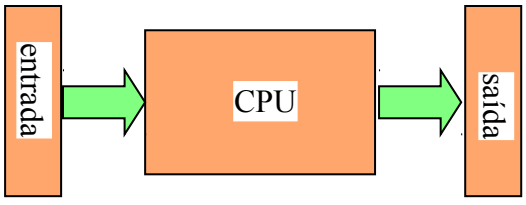


Figura 3.1 – Modelo do CLP

Os sinais de entrada e saída dos CLP's podem ser digitais ou analógicos. As entradas analógicas são módulos conversores A/D, que convertem um sinal de entrada em um valor digital. As saídas analógicas são módulos conversores D/A, ou seja, um valor binário é transformado em um sinal analógico.

Os sinais dos sensores são aplicados às entradas do controlador e a cada ciclo (varredura) todos esses sinais são lidos e transferidos para uma unidade de memória interna denominada memórias de imagem da entrada. Ao término do ciclo de varredura, os resultados são transferidos à memória imagem de saída e então aplicados aos terminais de saída – Figura 3.2.

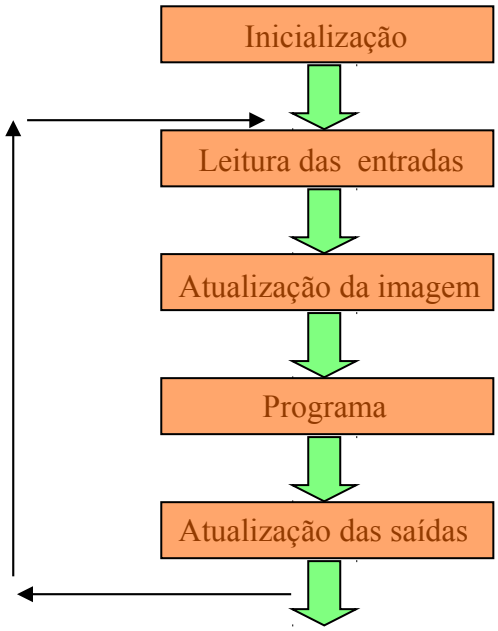


Figura 3.2 – Ciclo de execução do CLP

O CLP também pode se comunicar com outros equipamentos através de inúmeros protocolos de comunicação. Com o acoplamento de módulos específicos ele pode adquirir ou enviar dados via: Modbus-RS232, Modbus-RS485, Fieldbus entre outros.

3.3 – Entradas e Saídas

Como apresentado o CLP lê suas entradas e atualiza uma memória interna com estes valores. Como o escopo deste projeto é a validação da lógica de controle de operação de válvulas remotas, a seguir serão apresentadas as principais variáveis lidas deste equipamento no campo – Figura 3.3.

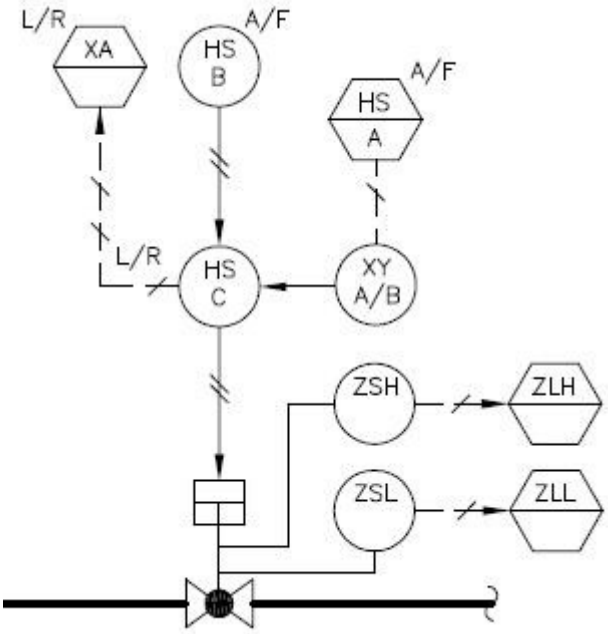


Figura 3.3 – Detalhe da sinalização da válvula

As chaves simbolizadas nos hexágonos – ZSL e ZSH - simbolizam as chaves de fim de curso da válvula, sendo estas normalmente fechadas. No momento em que a válvula abre a chave ZSH abre (NF) enquanto que quando a válvula fecha a chave ZSL é aberta (NF)

A variável simbolizada como L/R é uma chave física no campo e indica a operação local ou operação remota. Como a válvula pode ser comandada por dois locais diferentes (local e remotamente) é necessária esta chave para impedir a operação simultânea dos dois locais ao mesmo tempo.

O atuador indicado como A/F indica que a válvula indicada tem comandos de abrir e fechar. Sendo assim no momento em que uma saída do CLP é ativada este envia um sinal para o campo fazendo a válvula se movimentar.

No caso das chaves ZSH, ZSL e L/R ambas são contatos secos no campo que no momento em que abrem e fecham variam a entrada digital do CLP entre 0 e 1, a qual é utilizada na lógica interna (Figura – 3.4) – será apresentada no tópico seguinte a lógica de operação.

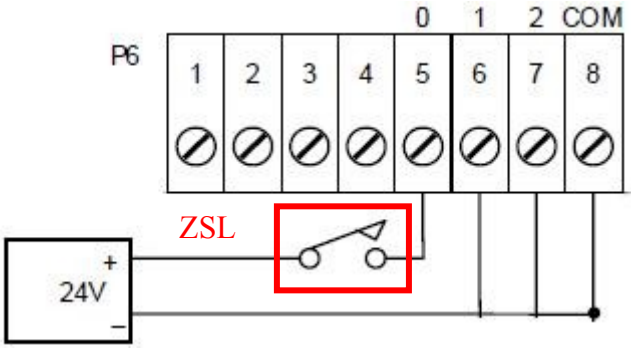


Figura 3.4 – Entrada digital no CLP – Exemplo ZSL
Fonte: Scadapack Light Manual [1]

No caso dos atuadores A/F geralmente um contato seco (relé) é fechado ou aberto no CLP interrompendo o sinal para o campo conforme a figura 3.5.

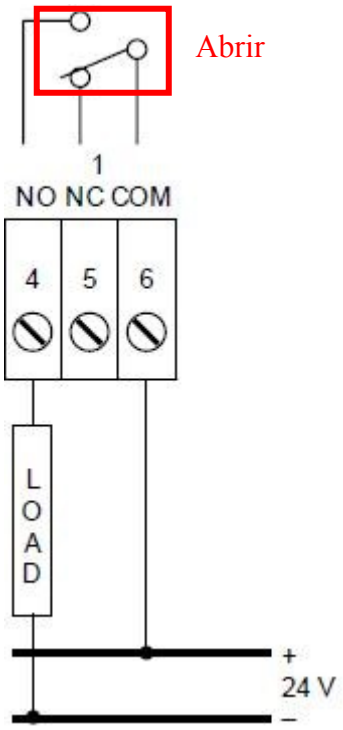


Figura 3.5 – Saída digital no CLP – Exemplo comando abrir
Fonte: Scadapack Light Manual [1]

3.4 – Diagrama Ladder

A lógica binária possui apenas dois valores que são representados por: 0 e 1. A partir desses dois símbolos construímos então uma base numérica binária. A partir desses conceitos foram criadas as portas lógicas, que são circuitos utilizados para combinar níveis lógicos digitais de formas específicas.

A linguagem Ladder permite que se desenvolvam lógicas combinacionais, seqüenciais e circuitos que envolvam ambas, utilizando como operadores para estas lógicas: entradas, saídas, estados auxiliares e registros numéricos. A Tabela 3.1 mostra os 3 principais símbolos de programação.

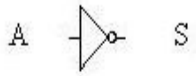
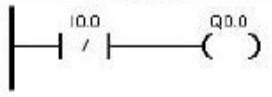
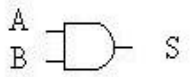
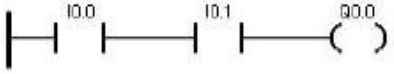
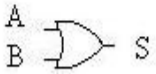
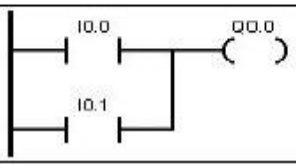
Portas Lógicas	Símbolo	Expressão	Ladder
NOT		$S = \overline{A}$	
AND		$S = A \cdot B$	
OR		$S = A + B$	

Figura 3.6 – Principais símbolos de programação ladder
Fonte: UERJ/LEE [2]

Mesmo tendo sido a primeira linguagem destinada especificamente à programação de CLP's, a linguagem ladder mantém-se ainda como a mais utilizada, estando presente praticamente em todos os CLP's disponíveis no mercado. O nome deve-se à representação da linguagem se parecer com uma escada (ladder), na qual duas barras verticais paralelas são interligadas pela Lógica de Controle, formando os degraus.

Cada símbolo da lógica de controle representa uma instrução da linguagem ladder sendo alocada em um endereço específico e consumindo uma quantidade determinada de memória disponível para armazenamento do programa de aplicação, conforme a CPU utilizada.

Na figura 3.4 está exemplificado o diagrama ladder no qual este projeto se baseou para validar a lógica das válvulas.

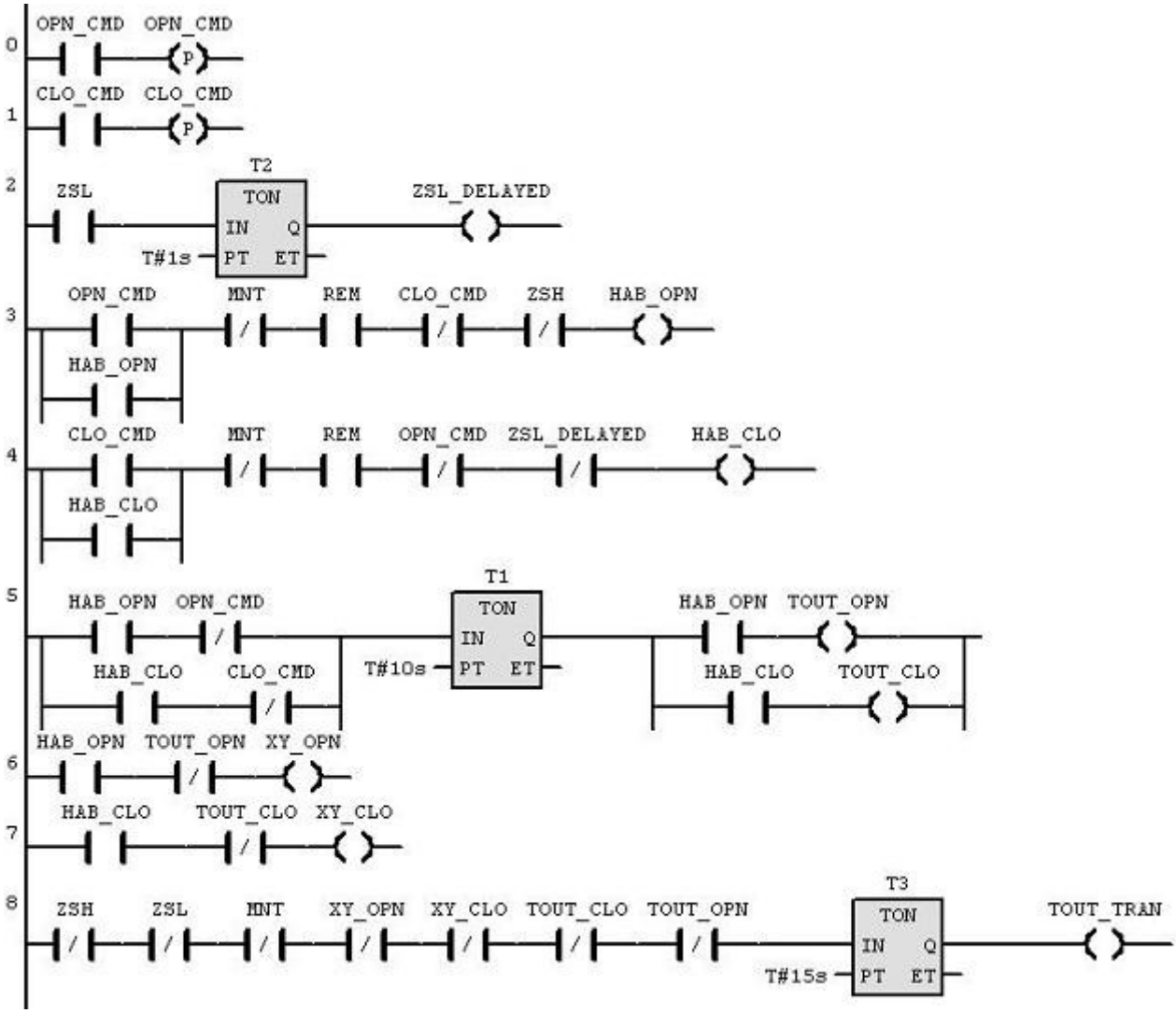


Figura 3.7 – Ladder da válvula sinalizada e comandada remotamente

Os símbolos ZSL, ZSH e REM estão diretamente ligados as suas respectivas entradas no CLP, desta forma assumindo o valor 0 ou 1 de acordo com sua condição de campo. Os contatos XY_OPN ou XY_CLO estão ligados, respectivamente, às saídas digitais dos comandos de abrir e de fechar. Os demais contatos são variáveis internas utilizadas para programação da lógica e para adequação ao padrão de operação da válvula.

As variáveis TOUT_CLO, TOUT_OPN e TOUT_TRAN são utilizadas para gerar a condição de time-out, a qual indica que a válvula não respondeu a um comando de abrir ou de fechar ou ainda permaneceu por um tempo determinado na condição de trânsito (não aberta e não fechada).

A condição MNT indica que a válvula está em manutenção, sendo está uma condição lógica. No momento em que um comando modbus para colocar a válvula em manutenção é dado somente é elevado um bit na palavra de saída do CLP indicando que a condição apresentada pela válvula não é confiável. No capítulo 7 será descrito todo o conjunto de testes juntamente com o perfil de funcionamento da lógica.

Todas a informações referentes à válvula são transferidas via modbus para fora do CLP através de uma palavra modbus, onde cada bit e a combinação deles tem um significado específico. No capítulo seguinte será apresentado mais detalhadamente o tratamento desta palavra e sua estrutura.

Capítulo 4

O Protocolo ModBus

4.1 – Introdução

O protocolo Modbus é um protocolo de mensagens da camada de aplicação, posicionado no nível 7 do modelo OSI. Foi desenvolvido pela Modicon Industrial Automation Systems fornecendo comunicação cliente / servidor entre dispositivos conectados em diferentes tipos de barramentos ou redes. Embora seja utilizado normalmente sobre conexões seriais padrão RS-232, ele também pode ser usado como um protocolo da camada de aplicação de redes industriais tal como TCP/IP sobre Ethernet. Este é, talvez, o protocolo de mais larga utilização em automação industrial, pela sua simplicidade e facilidade de implementação.

Durante as comunicações em uma rede Modbus, o protocolo determina como cada controlador vai saber o endereço do dispositivo, reconhecer uma mensagem dirigida a ele, determinar o tipo de ação a ser tomada, e extrair qualquer dados ou outras informações contidas na mensagem. Se a resposta for necessária, a controlador irá construir a mensagem de resposta e enviá-la usando o protocolo Modbus. Já em outras redes, as mensagens que contenham protocolo Modbus são encapsuladas no quadro ou na estrutura de pacotes que é utilizado na rede.

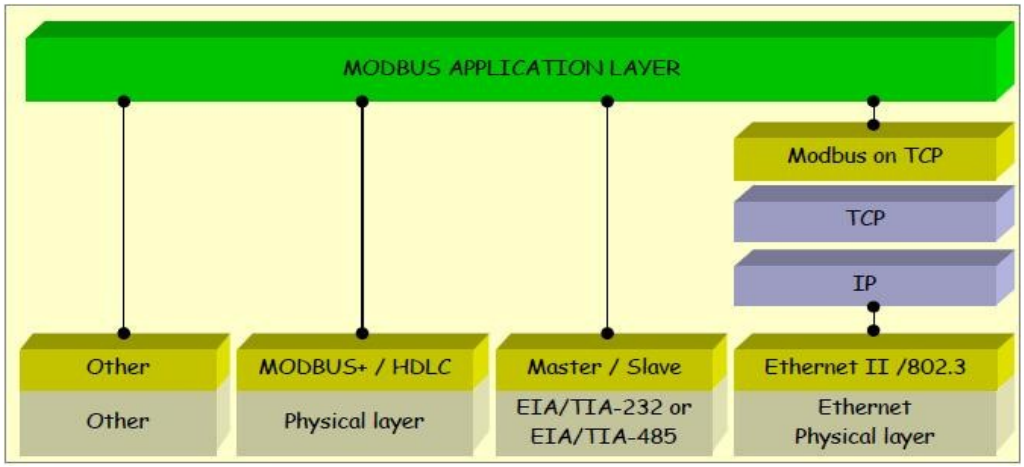


Figura 4.1 – Pilha modbus
Fonte: Modbus Application Protocol [3]

4.2 – Modelo de comunicação

O protocolo Modbus é baseado em um modelo de comunicação mestre-escravo, onde um único dispositivo, o mestre, pode iniciar transações denominadas *queries*. O demais dispositivos da rede (escravos) respondem, suprindo os dados requisitados pelo mestre ou executando uma ação por ele comandada (Figura 4.2). Geralmente o mestre é um sistema supervisório e os escravos são controladores lógicos programáveis. Os papéis de mestre e escravo são fixos, quando se utiliza comunicação serial, mas em outros tipos de rede, um dispositivo pode assumir ambos os papéis, embora não simultaneamente.

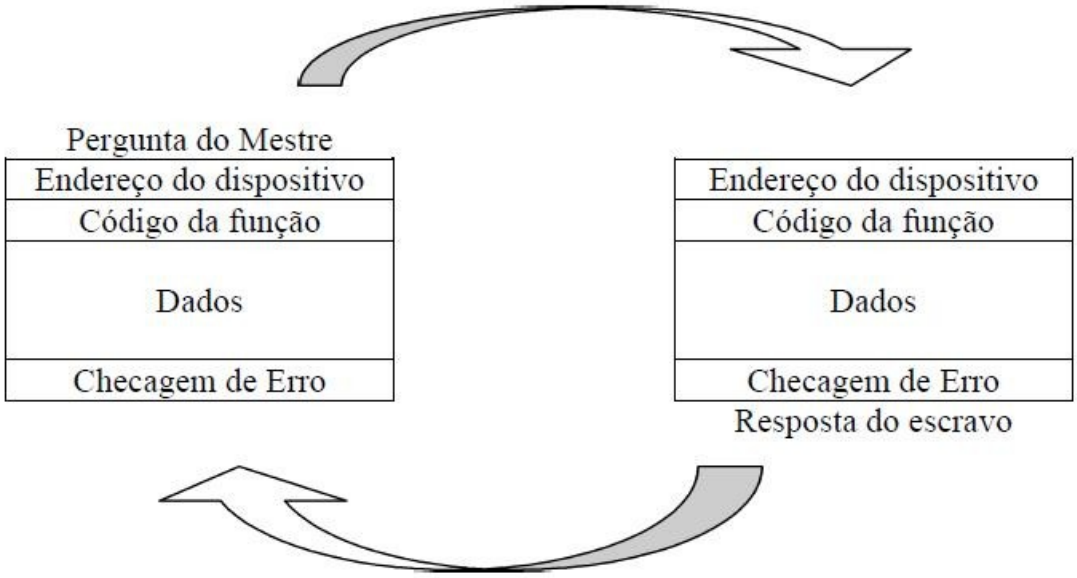


Figura 4.2 – Modelo de comunicação Modbus
Fonte: Protocolos orientados a caractere [4]

Query (pergunta do mestre ou request): O código de função na consulta informa aos dispositivos escravos destinatários que tipo de ação deve ser executada. Os bytes de dados contêm informações adicionais que o escravo vai precisar para executar a função. Por exemplo, o código de função 03 consulta o escravo para ler registros e responder com o seu conteúdo. O campo de dados deve conter as informações dizendo ao escravo o endereço do registro inicial e o número de registros que deverão ser lidos. O campo de verificação de erro fornece um método para o escravo validar a integridade do conteúdo da mensagem.

Response (resposta do escravo): Se o escravo faz uma resposta normal, o código da função na resposta é um eco do código de função na consulta. Os bytes de dados contêm as dados coletados pelos escravos, como os valores de registro ou de status. Se ocorrer um erro, o código de função é modificado para indicar que a resposta é uma resposta de erro, e os bytes de dados contêm um código que descreve o erro. O campo de verificação de erro permite que o mestre confirme se o conteúdo da mensagem é válido.

4.3 – Modos de Transmissão

Existem dois modos de transmissão: ASCII e RTU, que são selecionados durante a configuração dos parâmetros de comunicação. A seleção entre os modos refere-se apenas a redes padrão Modbus. Ele define o significado dos bits nos campos de mensagem transmitida serialmente nessas redes e determina como as informações serão empacotadas nos campos da mensagem e decodificada.

ASCII	Cada byte de mensagem é enviado como dois caracteres ASCII. Durante a transmissão, intervalos de até um segundo entre caracteres são permitidos, sem que a mensagem seja truncada. Algumas implementações fazem uso de tais intervalos de silêncio como delimitadores de fim de mensagem, em substituição à seqüência cr + lf.
Bits por Byte	1 – bit de início 7 – bits de dados (LSB enviado primeiro) 1 – bit de paridade; nenhum bit caso não tenha paridade 1 – bit de parada (se paridade habilitada); 2 – bits de parada (se paridade desabilitada)
Campo de Erro	LRC

Tabela 4.1 – Modo de transmissão ASCII

RTU	Cada byte de mensagem é enviado como um byte de dados. A mensagem deve ser transmitida de maneira contínua, já que pausas maiores que 1,5 caractere provocam truncamento da mesma. A principal vantagem deste modo em relação ao ASCII é que a sua maior densidade de dados que permite uma maior transferência de dados para a mesma taxa de transmissão.
Bits por Byte	1 – bit de início 8 – bits de dados (LSB enviado primeiro) 1 – bit de paridade; 1 – bit de parada ou 0 – bit de paridade; 2 – bits de parada
Campo de Erro	CRC

Tabela 4.2 – Modo de transmissão RTU

4.4 – Modelo de dados Modbus

O modelo de dados modbus é baseado em um conjunto de tabelas com suas características próprias. Estas tabelas formam um mapa de memória onde os dados podem ser lidos e/ou gravados. As quatro principais tabelas modbus estão representadas na tabela 4.3.

Tabela	Tipo	Permissão	Faixa Modbus (Endereços)
Discretes Input	1 bit	Somente Leitura	10001 - 19999
Coils	1 bit	Leitura/Escrita	1 - 1999
Input Registers	16 bits (palavra)	Somente Leitura	30001 -39999
Holding Registers	16 bits (palavra)	Leitura/Escrita	40001- 49999

Tabela 4.3 – Mapa de memória Modbus

. Para cada uma das tabelas primárias, o protocolo permite a seleção individual de 65.536 itens de dados dentro da faixa modbus, e as operações de leitura e escrita são projetadas para abranger múltiplos consecutivos de itens de dados até um limite de tamanho de dados que está dependente do código de função transação.

O endereço físico na memória não deve ser confundido com os endereços modbus. A única exigência é a referência de ligação de dados com endereço físico.

4.5 – Formato das mensagens (frame)

No modo ASCII, o frame começa com um 'dois pontos' (caractere ASCII 3A), e termina com um ‘retorno de carro - alimentação de linha’ (CR LF - ASCII 0D e 0A). Os caracteres permitidos para todos os outros campos são hexadecimal 0-9, A-F. Dispositivos de rede devem monitorar o barramento da rede continuamente para obter o caractere ‘dois pontos’. Quando um é recebido, cada dispositivo decodifica o campo seguinte (o endereço) para descobrir se é ele o dispositivo abordado. Intervalos de até um segundo podem decorrer entre os caracteres na mensagem. Se um intervalo maior ocorrer, o dispositivo receptor assume que um erro ocorreu. Um frame de mensagem ASCII típico é mostrado na tabela 4.4 abaixo.

Início	Endereço	Função	Dados	LRC	Fim
:	2 caracteres	2 caracteres	N caracteres	2 caracteres	CR LF

Tabela 4.4 – Frame ASCII

No modo RTU, o quadro começa com um intervalo de silêncio de pelo menos 3,5 vezes caracteres. O primeiro campo, em seguida, transmitido é o endereço do dispositivo.

Os caracteres permitidos para todos os campos são hexadecimal 0-9, A-F. Dispositivos de rede devem monitorar continuamente o tráfego, inclusive durante os períodos de "silêncio". Quando o primeiro campo (o campo de endereço) é recebido, cada dispositivo deve decodificá-lo para descobrir se ele é o dispositivo abordado. Na sequência do último caractere transmitido, um intervalo semelhante de pelo menos 3,5 vezes caracteres marca o fim da mensagem. Uma nova mensagem pode começar após esse intervalo.

O quadro de toda a mensagem deve ser transmitido como um fluxo contínuo. Se um intervalo de silêncio de 1,5 vezes caractere ocorre antes da conclusão do quadro, o dispositivo de recepção esvazia a mensagem incompleta e assume que o próximo byte será no campo de endereço de uma mensagem nova. Da mesma forma, se uma nova mensagem começar antes do silêncio, na sequência de uma mensagem anterior, o dispo-

sitivo de recepção vai considerá-lo uma continuação da mensagem anterior. Isto irá definir um erro, como o valor no campo CRC final não é válida para as mensagens combinadas. Um frame de mensagem RTU típico é mostrado na tabela 4.5 abaixo.

Início	Endereço	Função	Dados	LRC	Fim
Silêncio	8 bits	8 bits	N x 8 bits	16 bits	silêncio

Tabela 4.5 – Frame RTU

Campo Endereço: O campo de endereço de um quadro de mensagem contém dois caracteres (ASCII) ou oito bits (RTU). Os dispositivos escravos individuais têm endereços atribuídos no intervalo de 1 a 247. A mestre aborda um escravo, colocando o endereço do escravo no campo de endereço da mensagem. Quando o escravo envia a sua resposta, ele coloca o seu próprio endereço no campo de endereço de resposta para que o mestre sabe qual escravo está respondendo; O endereço 0 é utilizado para o endereço de broadcast, que todos os dispositivos escravos devem reconhecer.

Campo Função: O campo código de função de um quadro de mensagem contém dois caracteres (ASCII) ou oito bits (RTU), podendo assumir valores no intervalo de 1 a 255 (Figura 4.3). Quando uma mensagem é enviada de um mestre para um dispositivo escravo o campo código de função diz ao escravo que tipo de ação a ser executada. Exemplos disso são a ler estados digitais, grupos de estados digitais ou registros ou grupo de registros. Quando o escravo responde perante o mestre, ele usa o campo do código de função para indicar: uma resposta normal, neste caso o escravo simplesmente replica o código da função original; Ou um erro. O programa do dispositivo mestre tem a responsabilidade de lidar com erros.

				Function Codes		
				code	*Sub code*	*(hex)*
Data Access	**Bit access**	Physical Discrete Inputs	Read Discrete Inputs	**02**		**02**
		Internal Bits Or Physical coils	Read Coils	**01**		**01**
			Write Single Coil	**05**		**05**
			Write Multiple Coils	**15**		**0F**
	16 bits access	Physical Input Registers	Read Input Register	**04**		**04**
		Internal Registers Or Physical Output Registers	Read Holding Registers	**03**		**03**
			Write Single Register	**06**		**06**
			Write Multiple Registers	**16**		**10**
			Read/Write Multiple Registers	**23**		**17**
			Mask Write Register	**22**		**16**
			Read FIFO queue	**24**		**18**
	File record access		Read File record	**20**		**14**
			Write File record	**21**		**15**
Diagnostics			Read Exception status	**07**		**07**
			Diagnostic	**08**	**00-18,20**	**08**
			Get Com event counter	**11**		**OB**
			Get Com Event Log	**12**		**0C**
			Report Slave ID	**17**		**11**
			Read device Identification	**43**	**14**	**2B**
Other			Encapsulated Interface Transport	**43**	**13,14**	**2B**

Figura 4.3 – Mapa de funções Modbus
Fonte: Modbus Application Protocol [3]

Campo de Dados: O campo de dados é construído usando conjuntos de dois dígitos hexadecimais, no intervalo de FF a 00. Estes podem ser feitos a partir de um par de caracteres ASCII ou RTU, de acordo com o modo de transmissão da rede serial. O campo de dados de mensagens enviadas a partir de um mestre para um escravo contém informação adicional que o escravo deve usar para tomar as medidas definidas pelo código da função. Isso pode incluir itens como: a quantidade de itens a serem manuseados e; a contagem de bytes de dados reais no campo. Se não houver erro, o campo de dados de uma resposta de um escravo a um mestre contém os dados solicitados (Figura 4.4). Se ocorrer um erro, o campo contém um código de exceção que a aplicação do mestre pode usar para determinar a próxima ação a ser tomada (Figura 4.5).

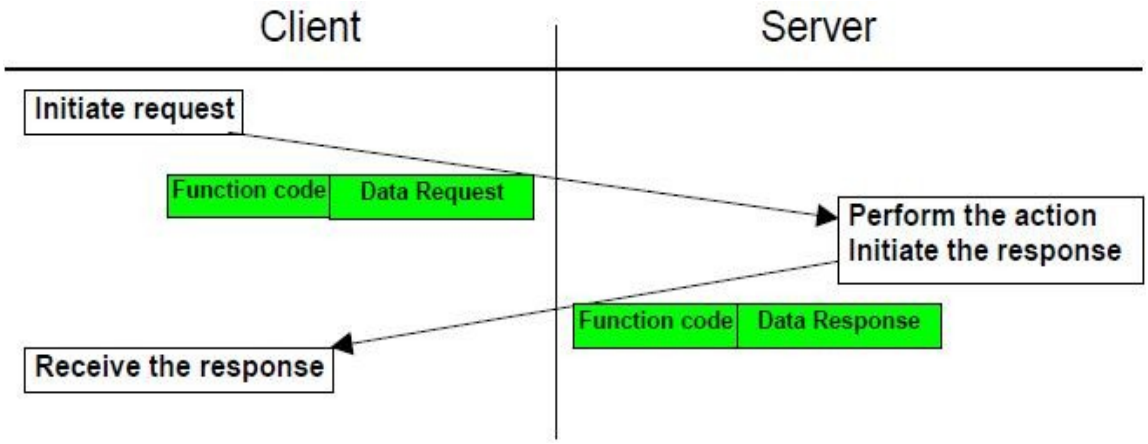


Figura 4.4 – Pergunta com resposta válida
Fonte: Modbus Application Protocol [3]

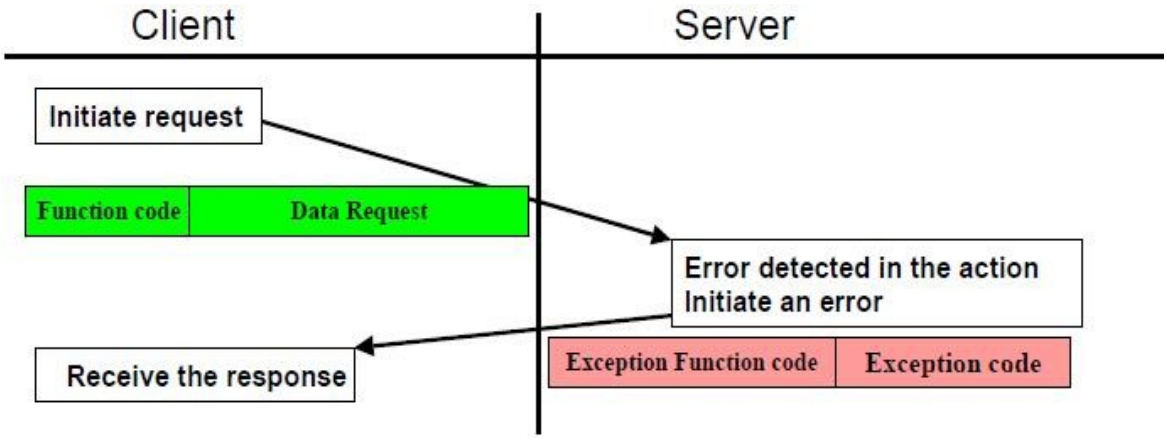


Figura 4.5 – Pergunta com resposta inválida
Fonte: Modbus Application Protocol [3]

Campo Checagem de Erro: Quando em modo ASCII o algoritmo verificador de erros utilizado é o LRC. Enquanto no modo RTU é utilizado o algoritmo CRC.

4.6 – API Jamod

Jamod é uma biblioteca escrita em Java utilizada para o desenvolvimento de aplicações modbus. Atualmente está na sua versão 1.2 e implementa todo o processo de montagem de quadros, transmissão, recepção e checagem de erros. Abordaremos aqui o princípio de funcionamento e sua arquitetura interna, visando familiarizar o leitor com as características do protocolo modbus apresentadas até agora. No capítulo 6, quando discutiremos o programa e seu funcionamento, serão detalhados de forma mais particular o relacionamento da biblioteca com a aplicação.

A Figura 4.6 seguinte ilustra o funcionamento básico da biblioteca e os componentes fundamentais de sua composição como, métodos, classes, variáveis utilizadas para o desenvolvimento deste projeto.

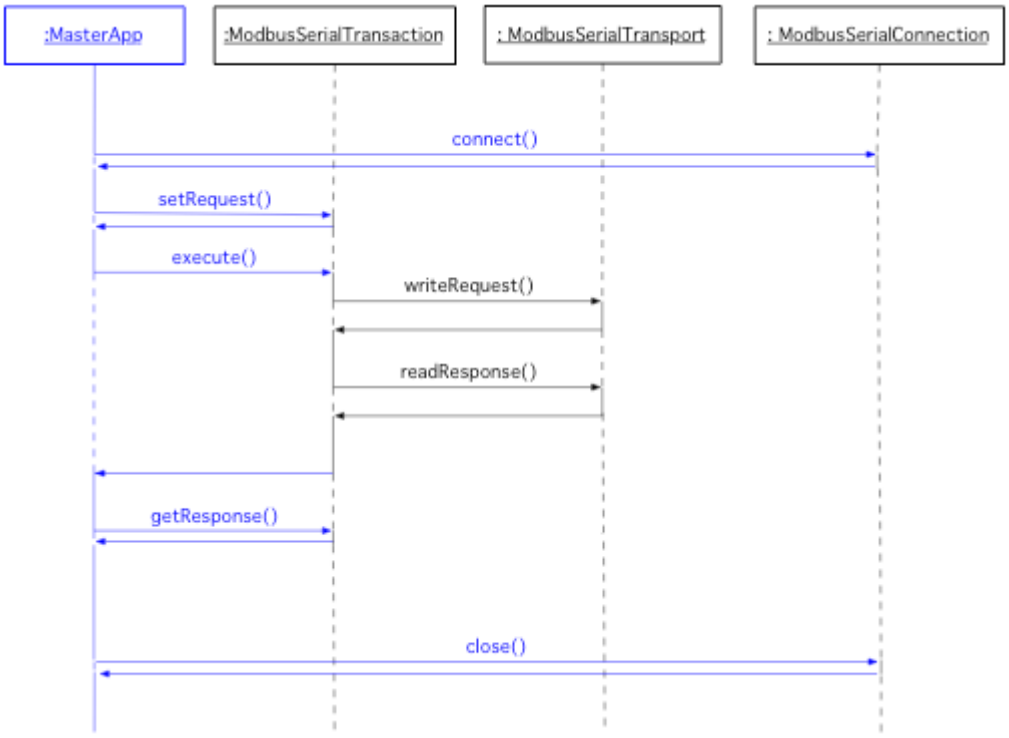


Figura 4.6 – Organização básica das classes da jamod
Fonte: Jamod Docs [7]

A classe ModbusSerialConnection implementa a parte física de conexão entre a aplicação mestre e o dispositivo escravo via serial (RTU ou ASCII) (via TCP/IP utiliza-se a ModBusTCPConnection) . Para este projeto foi utilizado ambos os tipos de conexão, visto que a grande maioria dos CLP's possui interface serial e em

contrapartida o “debug” do programa é mais facilmente implementado usando uma interface TCP/IP.

No momento em que o método connect é invocado, este se utiliza dos parâmetros de conexão previamente estabelecidos e estabelece um link entre o mestre (Aplicação) e o escravo. Após este momento é possível enviar perguntas e receber respostas dos escravos conectados a esta rede modbus. Para tanto se utiliza a classe ModbusSerialTransaction (via TCP ModbusTCPTransaction) em conjunto com a classe ModbusRequest.

Na classe ModbusRequest é inserido o endereço modbus a ser lido juntamente com o código de função criando assim um objeto desta classe. Com o método setRequest da classe ModbusSerialTransaction é passado o objeto ModbusRequest como parâmentro.

Assim que a aplicação principal chama o método execute da ModbusSerialTransaction, este chama o método writeRequest da classe ModBusSerialTransporte que é responsável por empacotar a pergunta no formato de codificação pré estabelecido e enviar ao escravo. Este mesmo objeto aguarda a resposta por um período de tempo pré-estabelecido (timeout) e retorna a resposta para o objeto ModbusSerialTransaction, o qual fica aguardando a aplicação mestre chamar o método getResponse.

O programa desenvolvido neste projeto faz uso em larga escala deste modelo, pois precisa monitorar as respostas vindas do CLP e também enviar novas requisições em determinados momentos.

4.7 – Comunicação com o CLP

Como já mencionado nos capítulos anteriores, a forma de comunicação entre um dispositivo e o CLP é geralmente através do protocolo modbus. As mensagens são enviadas tanto para adquirir valores de campo (armazenados na sua memória interna) ou para que estes realizem algum tipo de operação sobre a planta.

No caso da válvula, elemento de estudo deste projeto, a aplicação (ATS) envia uma mensagem para adquirir o estado da válvula em campo, enquanto o CLP responde enviando uma palavra (registro modbus – função 03) onde cada bit indica um estado

diferente para o instrumento. A seguir na tabela 4.6 é indicada a correspondência entre a posição dos bits dentro da palavra e seu significado.

bit	função
0	Local(0)/Remoto(1)
1	Automático(0)/Manual(1)
5	Aberta
6	Fechada
12	Time-out Abre
13	Time-out Fecha
14	Time-out Trânsito
15	Manutenção

Tabela 4.6 – Palavra de retorno modbus do CLP

No momento em que a aplicação recebe esta palavra ela pode usar os bits em nível 1 e animar a tela de um supervisório, por exemplo. Os bits 6 e 5 são utilizados em conjunto da maneira como mostrada na tabela 4.7.

Bit 5	Bit 6	Função
0	0	Trânsito
1	0	Aberta
0	1	Fechada
1	1	Falha

Tabela 4.7 – Combinação dos bits 5 e 6

Vale lembrar que o estado do bit não segue necessariamente a condição da chave no campo. Quando o bit 6 está atuado (em 1) e o bit 5 em 0 significa que a chave ZSH está aberta indicando 0 para o CLP e este convertendo para a lógica positiva. As chaves quando atuadas abrem para manter a condição de falha segura, na condição de rompimento do cabo, neste caso indicando falha para a aplicação.

Capítulo 5

O Protocolo ZigBee

5.1 – Introdução

Desenvolvido pela ZigBee Alliance junto ao IEEE o protocolo ZigBee é um padrão global e aberto para a comunicação sem fio de baixo custo, pouco consumo e pequeno alcance para utilização em diversas áreas desde automação industrial e residencial até equipamentos voltados para medicina.

A ZigBee Alliance é uma associação que conta com mais de 45 empresas, que trabalham em conjunto para desenvolver um padrão capaz de possibilitar um controle seguro, de baixo custo e potência em redes sem fio para o controle de diversos equipamentos. O protocolo suporta malha de roteamento da rede, permitindo que os pacotes de dados percorrerem vários nós a fim de alcançar o nó destino. Isso permite que nós ZigBee para se espalhem ao longo de uma grande região.

Nos tópicos seguintes serão apresentados de forma mais detalhada os conceitos fundamentais do protocolo.

5.2 – Dispositivos ZigBee

Podemos identificar dois tipos de dispositivos físicos em uma rede ZigBee, definidos pelo IEEE:

Full Function Device (FFD) - pode funcionar em toda a topologia do padrão, desempenhando a função de coordenador da rede e conseqüentemente ter acesso a todos os outros dispositivos. Trata-se de dispositivos de construção mais complexa.

Reduced Function Device (RFD) – é limitado a uma configuração com topologia em estrela, não podendo atuar como um coordenador da rede. Pode comunicar-se apenas com dispositivos FFD. São dispositivos de construção mais simples.

A tabela seguinte apresenta uma comparação entre os dispositivos de uma rede ZigBee com suas principais características:

FFD	RFD
Ajustes de parâmetros da rede	Função passiva na rede
Transmite informações pela rede	Efetua buscas por redes disponíveis
Gerencia os nós da rede	Transferência de dados da aplicação
Armazena informações dos nós de rede	Determina o status dos dados
Distribui mensagens entre nós de rede	Solicita dados ao coordenador da rede
Opera tipicamente no estado ativo	Pode permanecer no estado "sleep" por longos períodos

Tabela 5.1 – Principais características dos dispositivos ZigBee

O protocolo define três classes de dispositivos lógicos diferentes que formam uma rede ZigBee, com bases nos dispositivos físicos são eles : coordenador (FFD) , o roteador (FFD) e o dispositivo final (RFD) – Figura 5.1

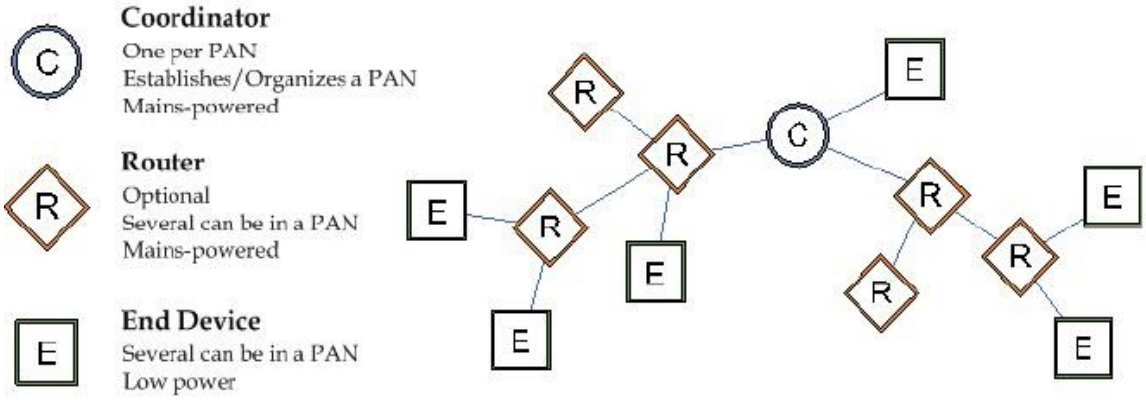


Figura 5.1 – Esquema típico de uma rede ZigBee
Fonte: Xbee manual – Zigbee protocol [8]

Coordenador - Responsável por selecionar o canal e PAN ID. O coordenador inicia um novo PAN; Uma vez que tenha começado um PAN, o coordenador pode permitir que os

roteadores e dispositivos finais a aderir ao PAN. O coordenador pode transmitir e receber transmissões de dados RF e pode auxiliar no encaminhamento de dados através da rede mesh. Como o coordenador deve ser capaz de permitir novas associações e encaminhar dados através de alguma rota, este deve ter uma alimentação que não seja por bateria.

Roteador - Um roteador deve participar de um PAN ZigBee antes que ele possa operar. Depois de juntar ao PAN, o roteador pode permitir que outros roteadores e dispositivos finais aderir ao PAN. O roteador também pode transmitir e receber RF transmissões de dados, e pode encaminhar pacotes de dados através da rede. Uma vez que os roteadores podem permitir novas adesões a rede e participar no encaminhamento de dados, os roteadores não podem entrar em modo 'sleep' (baixo consumo) e devem ter um alimentação mais duradoura .

Dispositivo Final - Um dispositivo final deve participar de um PAN ZigBee, semelhante a um roteador. O dispositivo final, no entanto, não pode permitir a adesão de outros dispositivos ao PAN, nem pode auxiliar no encaminhamento de dados através da rede. Um dispositivo final pode transmitir ou receber as transmissões de dados RF. Dispositivos finais são destinados ser alimentado à bateria. O dispositivo final pode podem entra em modo 'sleep', o roteador ou coordenador devem coletar todos os pacotes de dados destinados ao dispositivo final e armazenar em um buffer eles até o dispositivo acordar quando estará apto a recebê-los. O roteador ou coordenador que permitiu que o dispositivo final aderisse à rede deve gerenciar todos os dados de RF em nome do dispositivo final, por este motivo é conhecido como pai do dispositivo final. O dispositivo final é considerado um filho de seu pai.

5.3 – Topologia da Rede

A camada de rede ZigBee suporta três topologias de redes: estrela, árvore, e malha – Figura 5.2

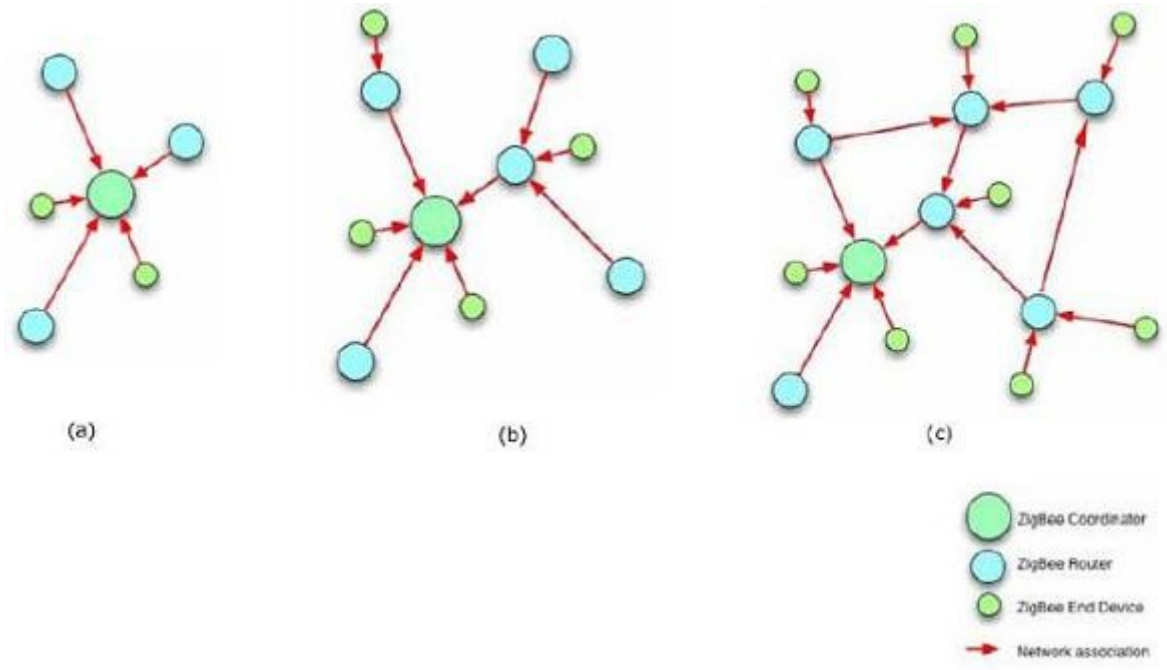


Figura 5.2 – Topologia de rede ZigBee; (a) Estrela; (b) Árvore; (c) Malha
Fonte: Xbee manual – Zigbee protocol [8]

5.4 – Iniciando uma rede ZigBee

O coordenador é responsável por iniciar uma rede ZigBee, toda rede deve ter um coordenador presente inicialmente. Para iniciar uma PAN, o coordenador realiza uma série de varreduras para descobrir o nível de atividade de RF em diferentes canais (energy scan – varredura energética) e para descobrir possíveis PAN's operacionais próximas (Pan Scan – varredura de PAN).

Quando um coordenador surge pela primeira vez, ele realiza uma varredura energética em múltiplos canais (frequências) para detectar os níveis de energia em cada canal. Canais com níveis de energia excessiva são removidos da lista de canais possíveis para iniciar a rede. A figura 5.3 ilustra este processo.

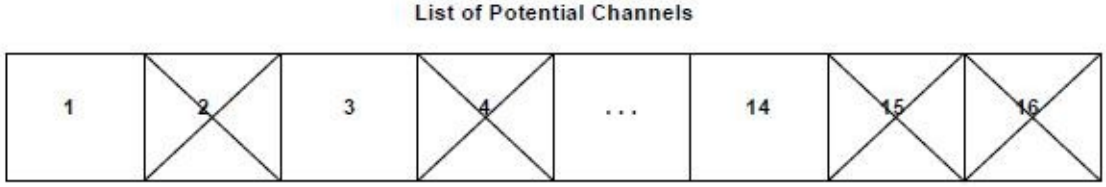


Figura 5.3 – Exemplo de canais potenciais para transmissão – 1, 3 e 14 – Energy Scan.
Fonte: Xbee manual – Zigbee protocol [8]

Quando o energy scan é concluído, o coordenador examina os canais potenciais para transmissão restantes por PAN's existentes. Para fazer isso, o coordenador envia um pacote do tipo 'beacon request' (Figura 5.4). Quaisquer coordenadores nas proximidades e roteadores irão responder ao pedido, enviando um pacote 'beacon response' de volta para o coordenador. O beacon contém informações sobre o PAN remetente, incluindo o identificador PAN ID, e se o dispositivo está permitindo novas adesões.

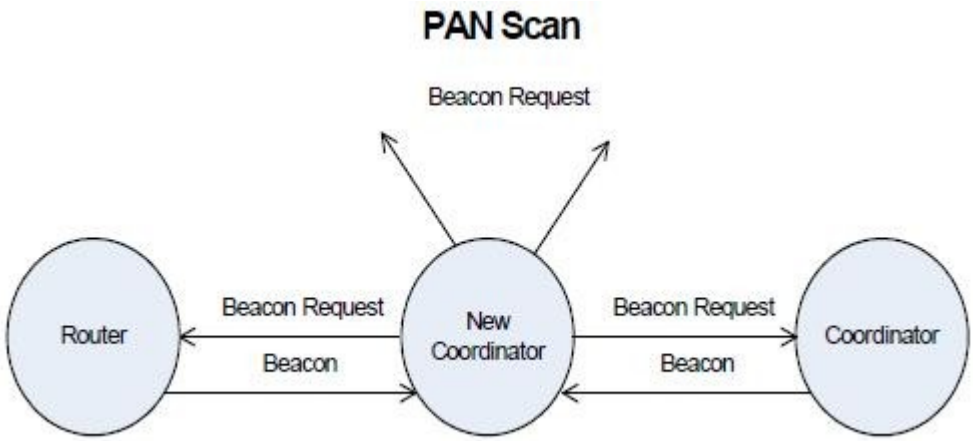


Figura 5.4 – Exemplo das perguntas e respostas aos pacotes beacon
Fonte: Xbee manual – Zigbee protocol [8]

Uma vez que o coordenador concluiu o PAN scan, ele analisa todos os pacotes beacon que recebeu para tentar de iniciar em uma rede com um PAN ID e um canal não utilizado. Quando o coordenador inicia um PAN, pode então permitir que os roteadores e / ou dispositivos finais adiram ao PAN.

5.5 – Aderindo à PAN

Roteadores dispositivos finais devem descobrir e aderir a uma PAN ZigBee. Para fazer isso, eles primeiro realizam um PAN scan, assim como o coordenador faz quando inicia uma PAN. Quando o PAN scan é realizado, o roteador ou dispositivo final recebe uma lista de pacotes beacon de dispositivos próximos. O dispositivo que está querendo entrar na rede então analisa esta lista para encontrar uma rede ZigBee válida para aderir.

Roteadores e dispositivos finais podem ser configurado para se juntar a qualquer PAN ZigBee, ou apenas para participar de uma rede com determinada PAN ID. No entanto, eles devem sempre encontrar um coordenador ou roteador que está permitindo novas adesões.Uma vez que um dispositivo de ligação (roteador ou dispositivo final) descobre uma PAN válida que está permitindo novas uniões, ele tenta se juntar ao PAN, enviando um pacote 'association request'. A figura 5.5 exemplifica este processo de adesões a PAN por novos dispositivos.

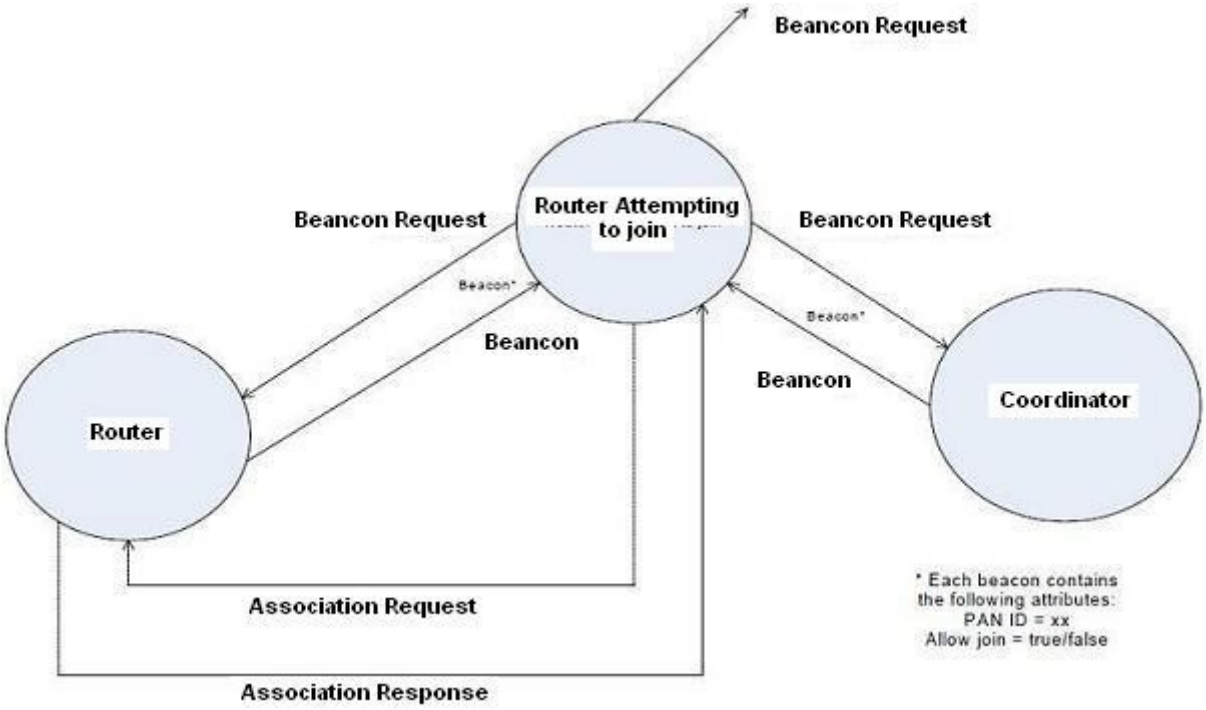


Figura 5.5 – Processo de adesão de novos dispositivos na rede
Fonte: Xbee manual – Zigbee protocol [8]

O coordenador e todos os roteadores podem permitir que os novos roteadores e dispositivos finais se juntar a eles. Um coordenador específico ou roteador irá permitir um novo dispositivo para participar depende de duas coisas: 1 – o atributo de permissão de novas adesões está habilitado; 2 - O número de dispositivos filhos já excedeu.

Atributo Permit-Joining: O coordenador e todos os roteadores têm um atributo permit-joining. Este atributo em um coordenador e qualquer roteador pode ser configurado

para permitir sempre novas adesões, permitir novas adesões em um curto período de tempo, ou para não permitir mais nenhuma adesão.

Dispositivos filhos: O coordenador e cada roteador podem suportar um número finito de filhos. Uma vez que o número de dispositivos finais juntou-se um determinado roteador ou coordenador, o dispositivo não pode mais novas adesões.

5.6 – Endereçamento físico

O protocolo 802.15.4 onde o protocolo ZigBee é construído especifica dois tipos de endereço:

Endereço 16-bit: É um endereço de rede único atribuído a cada nó quando se une a uma PAN. No entanto, os endereços de rede não são estáticos - podem alterar. Uma das duas condições seguintes validará um nó a receber um novo endereço de rede: 1 – Se um dispositivo final não pode se comunicar com seu pai ele pode precisar sair da rede e voltar para encontrar um novo pai; 2. Se houver alterações do tipo roteador para dispositivo final ou vice-versa, o dispositivo irá deixar a rede e voltar como um dispositivo novo. O protocolo requer que os dados sejam enviados para um destino através de seu endereço 16-bits. Para tanto o endereço precisa ser descoberto antes de transmitir dados. No tópico 5.8 será apresentado o modo de roteamento do protocolo.

Endereço 64-bit: Cada nó contém um endereço único de 64 bits. Este identifica um nó e é permanente.

5.7 – Endereçamento da camada de aplicação

A camada de aplicação ZigBee define “cluster ID’s” (cluster identifiers) e “endpoints” que são usados para endereçar serviços individuais ou aplicativos em um dispositivo. Um endpoint é uma tarefa ou aplicação que é executado em um dispositivo Zig-

Bee, semelhante a uma porta TCP. Cada dispositivo ZigBee pode suportar um ou mais endpoints. Os clusters ID definem uma determinada função ou ação em um dispositivo.

Na figura 5.6 é apresentados um esquema típico de cluster e endpoint; o cluster é o responsável por executar determinada tarefa em um certo endpoint. Desta forma pode-se ter um único cluster ID e ele desempenhar a mesma função em inúmeros endpoints.

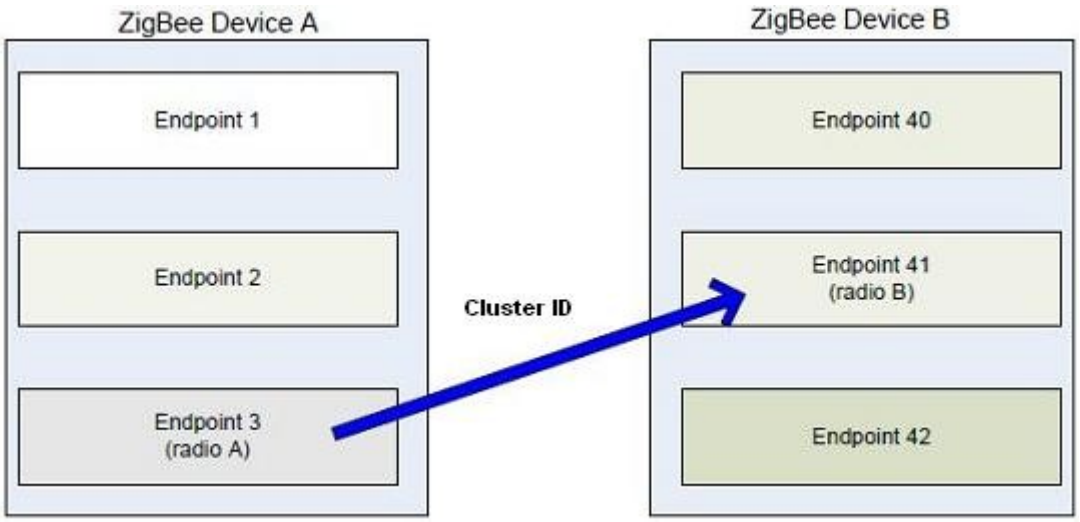


Figura 5.6 – Representação dos modelos de endpoint e cluster ID
Fonte: Xbee manual – Zigbee protocol [8]

5.8 – Tipos de transmissão

Existem 2 tipos de transmissão em uma rede ZigBee:

Broadcast: Destinam a ser propagada em todo a rede de tal forma que todos os nós recebam a transmissão. Para isso, todos os dispositivos que recebem uma transmissão do tipo broadcast irá retransmitir o pacote 3 vezes. Cada nó que transmite um pacote broadcast irá também criar uma entrada em uma tabela de transmissão broadcast. Esta entrada é utilizada para manter o controle de cada pacote broadcast recebido para assegurar que os pacotes não sejam infinitamente transmitidos. A tabela de transmissão broadcast detém 8 entradas persistentes por 8 segundos. Para cada transmissão broadcast, a pilha ZigBee deve reservar espaço em memória para uma cópia do pacote

de dados. Essa cópia é usada para retransmitir o pacote caso necessário. A figura 5.7 exemplifica uma transmissão broadcast.

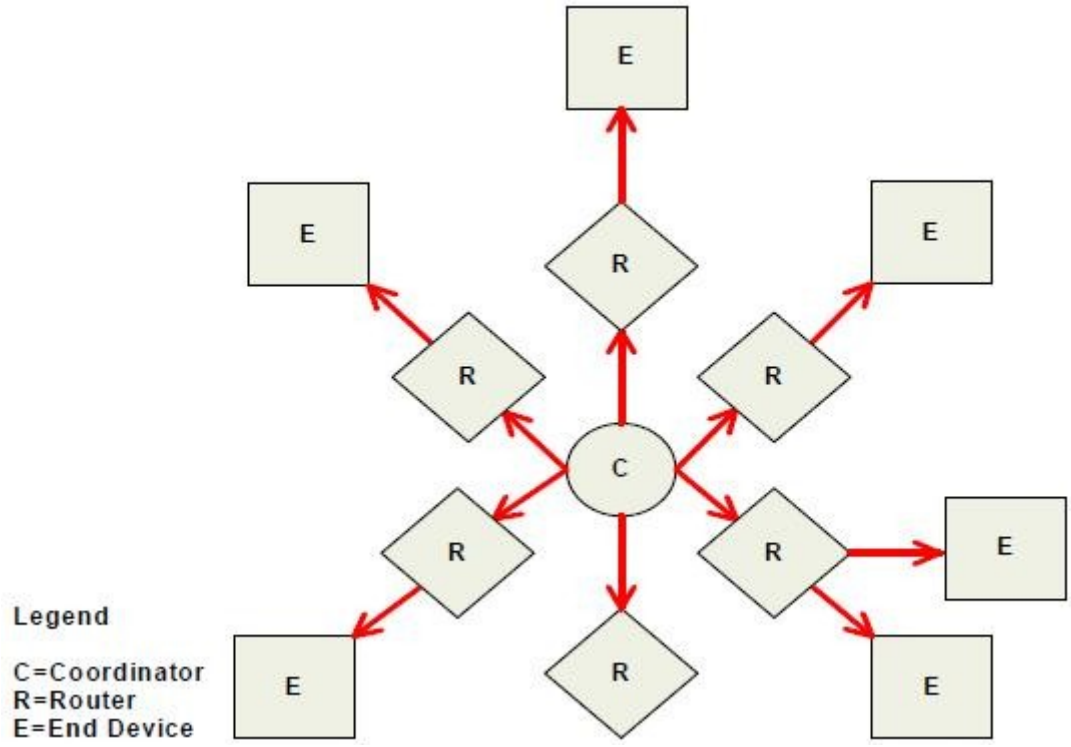


Figura 5.7 – Pacotes broadcast inundando a rede ZigBee
Fonte: Xbee manual – Zigbee protocol [8]

Unidcast: Transmissões deste tipo são sempre dirigidas a um endereço 16-bit do dispositivo, que não é estático. No entanto, apenas o endereço 64-bit de um dispositivo é permanente, portanto dispositivos ZigBee podem empregar descoberta de endereços(Network Address Discovery – NAD) para identificar o endereço 16 bits que corresponde a um endereço 64-bits conhecido, e de descoberta de rota (Route Discovery – RD) para estabelecer uma rota até o dispositivo – Figura 5.8.

5.9 – Roteamento

Nas transmissões do tipo unicast, conforme foi mencionado o protocolo precisa fazer o uso de duas ferramentas para estabelecer a comunicação entre dois dispositivos:

Network Address Discovery - NAD: Para descobrir o endereço de rede que foi atribuído a um determinado dispositivo quando ele se juntou ao PAN o dispositivo iniciador inicia uma transmissão broadcast de descoberta de endereços através da rede. Este pacote contém o endereço de 64 bits do dispositivo que o iniciador precisa enviar dados. Os dispositivos que recebem este pacote verificam se seus endereços de 64-bits coincidem com o endereço de 64 bits contido na transmissão broadcast. Caso afirmativo, o dispositivo envia um pacote de resposta para o iniciador, fornecendo o endereço de rede (16 bits) do dispositivo com o endereço 64-bits correspondente. Quando esta resposta é recebida, o iniciador pode transmitir dados.

Route Discovery – RD: O roteamento "mesh" é utilizado para estabelecer uma rota entre o dispositivo de origem e de destino. Este algoritmo permite que os pacotes de dados percorram vários nós, inclusive roteadores e coordenadores, em uma rede para encaminhar os dados de uma fonte para um destino. O route Discovery baseia-se no algoritmo de roteamento AODV (Ad-hoc On-demand Distance Vector).

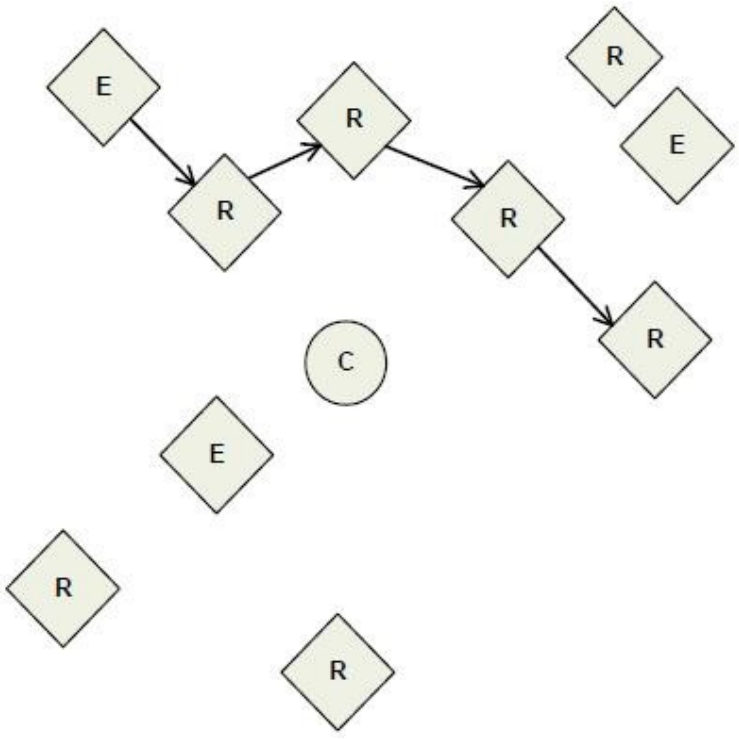


Figura 5.8 – Transmissão unicast através da rede mesh
Fonte: Xbee manual – Zigbee protocol [8]

Roteamento sobre o algoritmo AODV é feito usando tabelas em cada nó que registra o próximo nó (nós intermediários entre a origem e o destino) para um nó destino (Exemplificado na tabela 5.2). Se o próximo nó não é conhecido, o route discovery deve ser acionado a fim de encontrar um caminho. Uma vez que apenas um número limitado de rotas podem ser armazenados em um roteador, a descoberta de rota será realizada com mais freqüência em uma rede de comunicação com muitos nós diferentes.

Nó	Endereço destino	Endereço próximo nó
R3	Router 6	Coordinator
C	Router 6	Router 5
R5	Router 6	Router 6

Tabela 5.2 – Tabela de roteamento

Quando um nó de origem precisa descobrir uma rota para um nó de destino, ele envia um pacote broadcast com um comando de solicitação de rota. Este comando contém o endereço de rede de origem, o endereço de rede destino e um campo de custo de caminho (uma métrica para medir a qualidade da rota). Como o comando de solicitação é propagado através de toda rede, cada nó que re-transmite a mensagem atualiza o campo de custo de caminho e cria uma entrada temporária em sua tabela de descoberta de rota. Na figura 5.9 o nó R3 está tentando descobrir uma rota para o dispositivo R6.

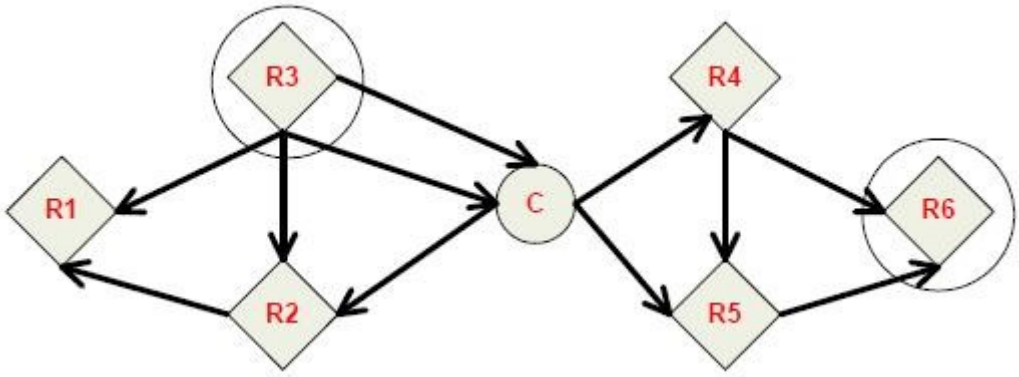


Figura 5.9 – Transmissão broadcast com descoberta de rota.
Fonte: Xbee manual – Zigbee protocol [8]

Quando o nó destino recebe um pedido de rota, ele compara o campo custo de caminho com os pedidos de rota recebidos anteriormente. Se o custo de caminho contido no pedido de rota é melhor do que qualquer outra anteriormente recebida, o nó de destino irá transmitir um pacote de resposta de rota para o nó que originou o pedido de rota. Nós intermediários recebem e transmitem o pacote de resposta de rota para o nó de origem (o nó que originou solicitação). A figura 5.10 ilustra o processo de resposta de um dispositivo o processo de route Discovery, note que o dispositvo de destino pode transmitir múltiplas respostas caso este descubra rotas melhores.

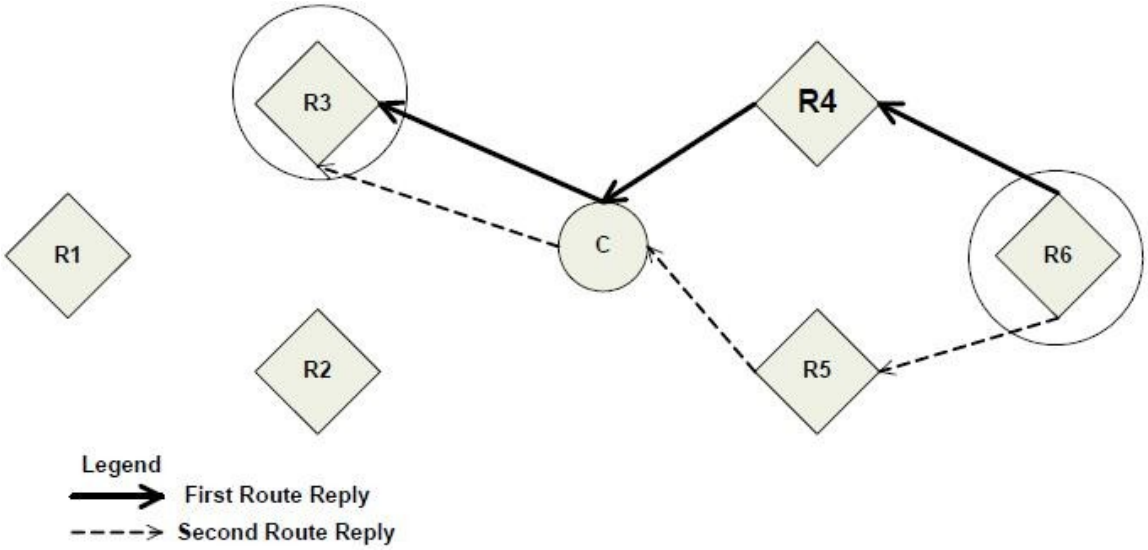


Figura 5.10 – Resposta de uma requisição de rota.
Fonte: Xbee manual – Zigbee protocol [8]

O protocolo também define pacotes de reconhecimento, tanto no nível físico quanto camada de aplicação. Como os dados são transmitidos de um nó para seu vizinho, um pacote de confirmação (ACK) é transmitido na direção oposta, para indicar que a transmissão foi recebida com êxito. Se o ACK não é recebido, o dispositivo irá retransmitir os dados, até 4 vezes. Este ACK é chamado de reconhecimento a nível físico ou reconhecimento da camada MAC. Além disso, o dispositivo que originou a transmissão aguarda um pacote (ACK) do dispositivo de destino. Esta ACK vai percorrer o mesmo caminho que os dados percorreram, mas no sentido oposto. Caso o pacote ACK não for recebido, ele irá retransmitir os dados, até 2 vezes, até que um ACK seja recebido. Este ACK é chamado de reconhecimento da camada APS. Na

figura 5.11 é ilustrado o comportamento dos pacotes de dados, juntamente com os pacotes de reconhecimento MAC e de aplicação em uma determinada rota.

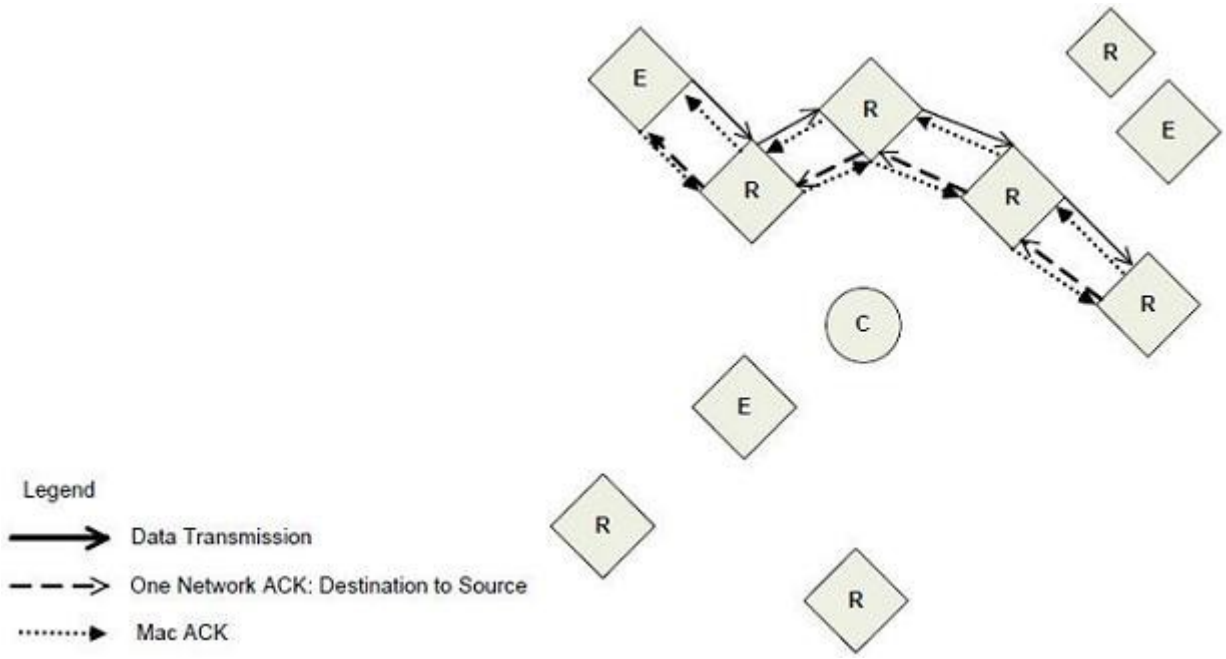


Figura 5.11 – Exemplificação dos pacotes de reconhecimento.
Fonte: Xbee manual – Zigbee protocol [8]

5.10 – Pilha de protocolos

A arquitetura ZigBee é composta de um conjunto de blocos, chamados de camadas (figura – 5.12). Cada camada realiza um conjunto específico de serviços para a camada acima. Cada entidade é uma função da camada, como por exemplo, a entidade de dados fornece um serviço de transmissão de dados e uma entidade de gerenciamento fornece todos os outros serviços. Cada entidade de serviço expõe uma interface para a camada superior através de um serviço do tipo ponto de acesso (SAP) suportando um número de primitivas de serviço para atingir a funcionalidade necessária.

A norma IEEE 802.15.4 define as duas camadas inferiores: a física (PHY) e o controle de acesso ao meio (MAC). A camada PHY pode operar em duas frequências separadas: 868/915 MHz e 2,4 GHz. A menor freqüência camada PHY abrange tanto a 868 MHz (Européia) e a 915 MHz, utilizado em países como os Estados Unidos e Austrália. A maior freqüência de camada PHY é usado praticamente todo o mundo. A ca-

mada MAC controla o acesso ao canal de rádio usando um mecanismo CSMA-CA. Entre suas responsabilidades podem também incluir a transmissão pacotes beacon, sincronização proporcionando uma transmissão confiável.

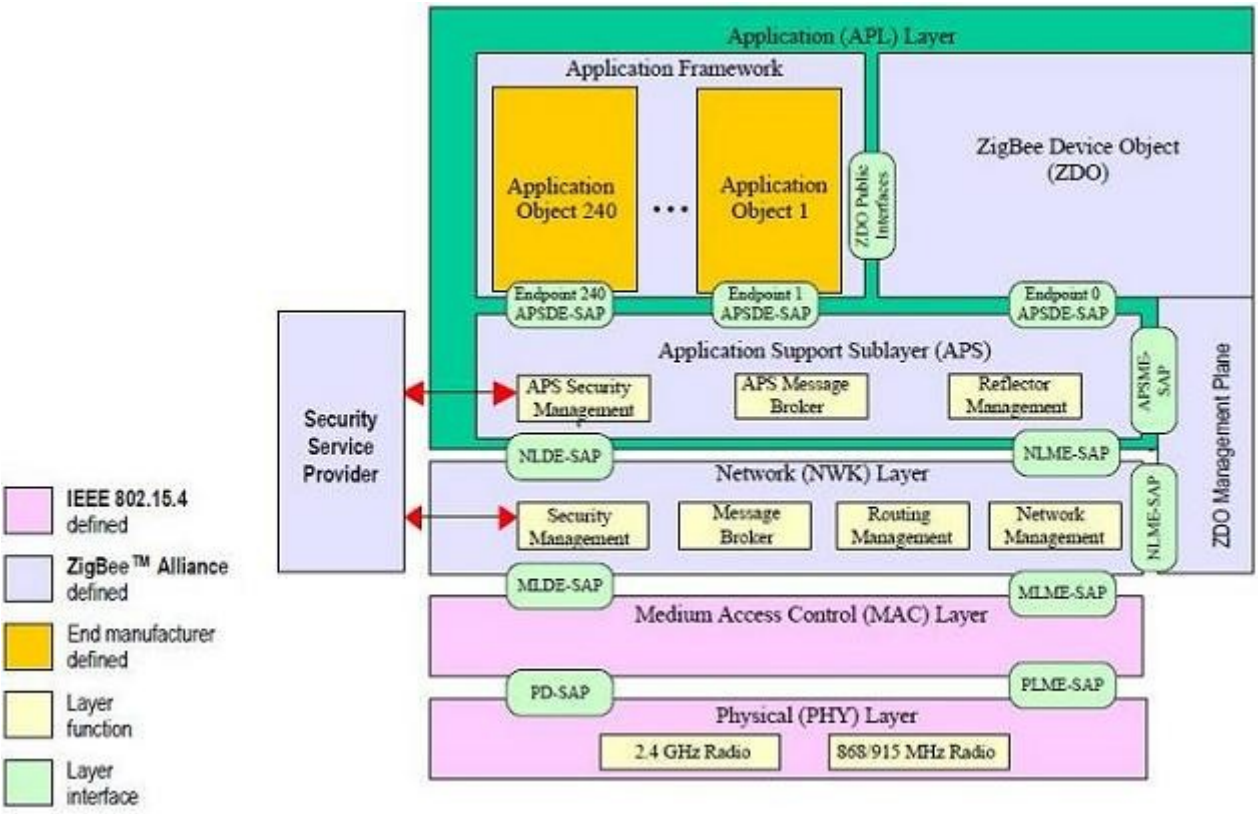


Figura 5.12 – Pilha de protocolos ZigBee
Fonte: ZigBee Protocol Specification [9]

O detalhamento de cada camada foge do escopo deste projeto, visto que é a implementação do protocolo. Mais informações podem ser encontradas em [9].

Capítulo 6

Módulo Xbee

6.1 – Introdução

Os módulos XBee's são dispositivos RF projetados para operam no âmbito do protocolo ZigBee e fundamentam sua premissa de necessidades de baixo custo, baixa potência e proporcionar confiança na entrega de dados entre dispositivos remotos. Operando na freqüência de 2,4 GHz ISM (industrial, scientific and medical), cuja faixa é mundialmente reservada para aplicações RF, garante interoperabilidade entre diversos equipamentos fabricados em diversas partes do mundo.

No decorrer deste capítulo serão apresentados como é realizada a comunicação entre um dispositivo externo e o módulo, suas características intrínsecas e o detalhamento da comunicação com um dispositivo remoto.

Na figura 6.1 é apresentada uma visão do módulo e seus pinos de entradas e saídas usados para controle, comunicação e diagnóstico do módulo.

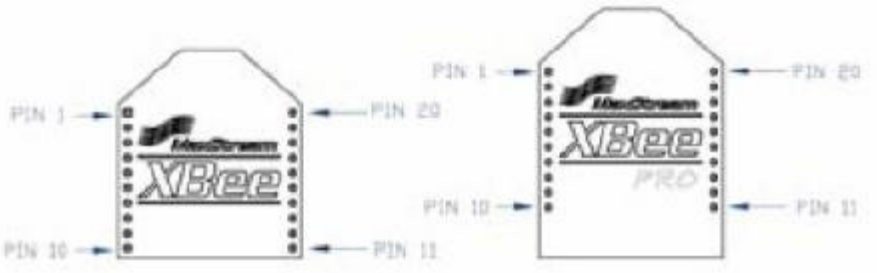


Figura 6.1 – Visão do módulo Xbee
Fonte: Xbee manual – Zigbee protocol [8]

6.2 – Estrutura de comunicação serial

O XBee se comunica com um outro dispositivo de hardware através de uma comunicação serial assíncrona (RS-232) – Figura 6.2

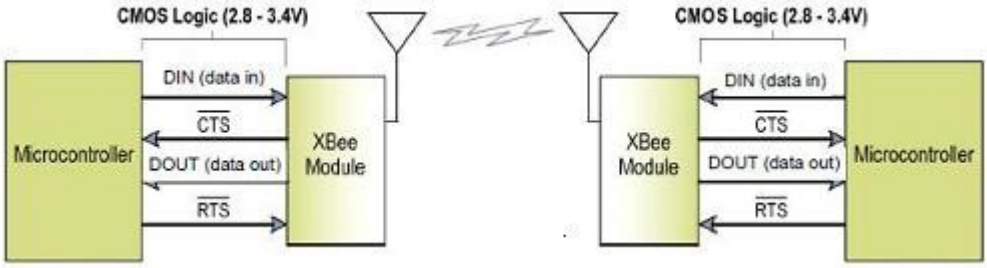


Figura 6.2 – Exemplo de comunicação entre um módulo e um microcontrolador
Fonte: Xbee manual – Zigbee protocol [8]

Os módulos mantêm pequenos buffers para coletar os dados via RF e via serial na ordem recebida, exemplificado na figura 6.3. O buffer serial de recepção (Serial Receiver Buffer) coleta dados da entrada (pino sete dispositivo – DIN) até que possam ser processados. O buffer serial de transmissão (Serial Transmit Buffer) recolhe dados recebidos via RF e posteriormente serão transmitidos para fora do módulo (pino 8 dispositivo – DOUT) – Figura 6.3. O RF Switch define em quem terá acesso ao canal de transmissão podendo alternar entre transmitir dados ou receber dados de acordo com as condições dos buffers.

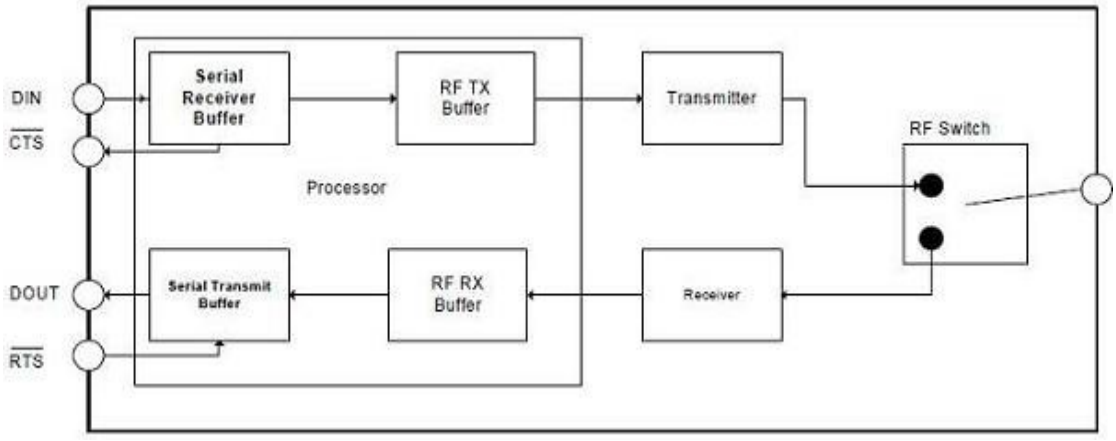


Figura 6.3 – Modelos de comunicação com o módulo Xbee
Fonte: Xbee manual – Zigbee protocol [8]

Para dar início transmissão serial entre o Xbee e um dispositivo de hardware o sinal deve estar em Idle (nível lógico alto) quando não há dados a serem transmitidos. Cada byte de dados consiste em um bit de inicío (start bit) em nível baixo, oito bits de dados (LSB primeiro) e um bit de parada (stop bit) em nível lógico alto. A figura 6.4 a seguir ilustra o padrão de bits em uma linha serial RS-232.

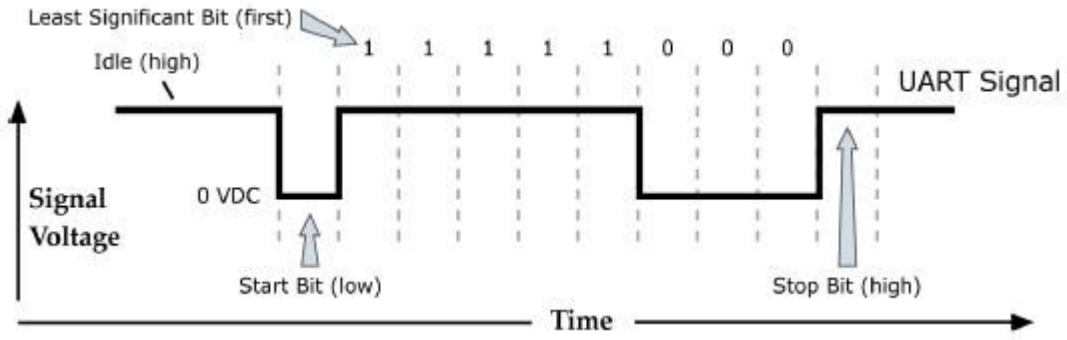


Figura 6.4 – Padrão de comunicação 8-N-1 (dados – paridade – # bits de parada)
Fonte: Xbee manual – Zigbee protocol [8]

Existem dois protocolos seriais sobre RS-232, que são passíveis de comunicação com o módulo: modo Transparente e modo API (Application Programming Interface); A comutação do dispositivo entre os modos só é possível através da atualização de firmware. Uma vez o módulo operando por um determinado protocolo serial todos os elementos da rede devem estar configurados para o mesmo modo. Nos capítulos seguintes serão discutidos com mais detalhes estes protocolos.

6.3 – Modo Transparente

Quando configurados para operarem neste modo, os módulos agem como um substituto de uma linha serial cabeada comum. Todos os dados recebidos através do pino DIN entram na fila de transmissão RF. Quando os dados via RF são recebidos, estes são enviados através do pino DOUT. Dessa forma o módulo implementa uma forma de linha serial wireless.

Os dados são colocados no Serial Receiver Buffer até que uma das seguintes condições determine que os dados sejam empacotados e transmitidos:

- Nenhum dado serial é recebido em um período de tempo determinado pelo parâmetro RO (Packetization Timeout). Se RO = 0, o dado é imediatamente empacotado e enviado via RF.
- Número máximo de dados que contém em um pacote RF é recebido (72 bytes).
- A sequência de comandos (GT + CC + GT) é recebida.

Os parâmetros de configuração do módulo são configurados através do modo de comandos de interface AT. Para modificar ou ler os parâmetros do módulo Xbee, este deve primeiro entrar no command mode (modo de comando) – estado no qual os dados recebidos são interpretados como comandos.

Para entrar no modo de comando a seguinte sequência deve ser obedecida:

- Nenhum dado deve ser enviado por um tempo predeterminado pelo parâmentro GT (Tempo de Guarda) – Default um segundo.
- Enviar três caracteres “+” em sequência – "+++" – dentro do intervalo de um segundo. [CC (Command Sequence Character) – define o caractere de sequência].
- Nenhum dado deve ser enviado por um tempo predeterminado pelo parâmentro GT (Tempo de Guarda) – Default um segundo.

Uma vez que o módulo entra em no modo de comando AT, este envia a sequência de caracteres "OK \ r" através do pino DOUT e o temporizador do modo de comando é iniciado (parâmetro CT). Desta forma todos os parâmetros de configuração podem ser modificados para atenderem as necessidades da aplicação. Na figura 6.5 está exemplificado um comando AT para alterar o endereço de destino no dispositivo para o valor 1F. É necessário para que os dados sejam gravados na memória flash (não volátil) um comando WR (Write command).

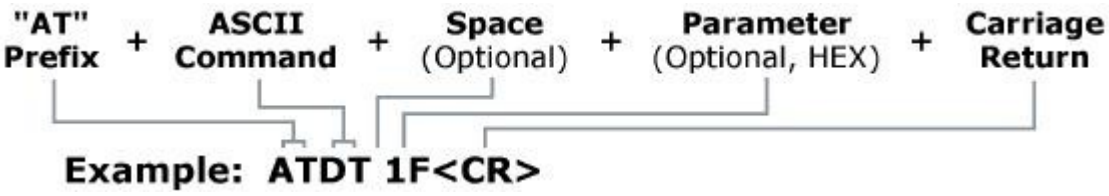


Figura 6.5 – Sintaxe de um comando em modo transparente
Fonte: Xbee manual – Zigbee protocol [8]

Todo comando enviado ao dispositivo e efetuado com sucesso é retornado à aplicação uma sequência "OK \r" e caso haja algum erro a sequência "ERRO" é retornada. Para que os comandos efetuados com sucesso tenham efeito na operação do módulo é necessário que o comando AC (Apply Changes) seja enviado ou o modo comando seja finalizado.

6.4 – Modo API

Como uma alternativa para o funcionamento em modo transparente, existe o protocolo serial API (Application Programming Interface). Comandos, respostas de comandos e mensagens de status são enviados e recebidos do módulo usando um UART Data Frame (Quadro de Dados Serial).

Este projeto utiliza-se desta capacidade do módulo, visto que é mais abrangente, robusta e existe uma biblioteca aberta que implementa os quadros de comunicação. Uma das principais vantagens deste modo de comunicação é a capacidade de poder alterar dispositivos remotos somente via RF. Como será apresentado este modelo se baseia no conceito de "endpoints", apresentado no capítulo anterior, podendo realizar operações sobre o módulo sem a necessidade de um dispositivo externo ligado serialmente.

Dois modos API estão disponíveis através do parâmetro AP (API Enable): AP = 1 – Modo API ativado; AP = 2 – Modo API ativado com caracteres de escape. Alguns caracteres precisam ter um sequência de escape para que não sejam interpretados de maneira errada. A tabela 6.1 identifica os caracteres que precisam de um sequência de escape.

Caractere (Hexa)	Função
7E	Delimitador de quadro
7D	Escape
11	XON
13	XOFF

Tabela 6.1 – Caracteres que precisam da sequência de escape

Exemplo de quadro com um caractere a ser escapado.

Sequência original: 0x7E 0x00 0x02 0x23 0x11 0xCB
Caractere a ser escapado: 0x11

Sequência final: 0x7E 0x00 0x02 0x23 0x7D 0x31 0xCB

O caractere a ser escapado é combinado logicamente com o caractere 0x20 através da função XOR e precedido do caractere 0x7D.

Um modelo típico de quadro API é apresentado na figura 6.6. Todo quadro começa com um delimitador de quadro de 1 byte. A seguir 2 ou 3 bytes identificam o tamanho do quadro. O quadro de dados é composto pelo tipo de quadro e os dados específicos de cada quadro, podendo ter tamanho variável. O algoritmo de verificação de erro utilizado é o CLC e ocupa 1 byte.

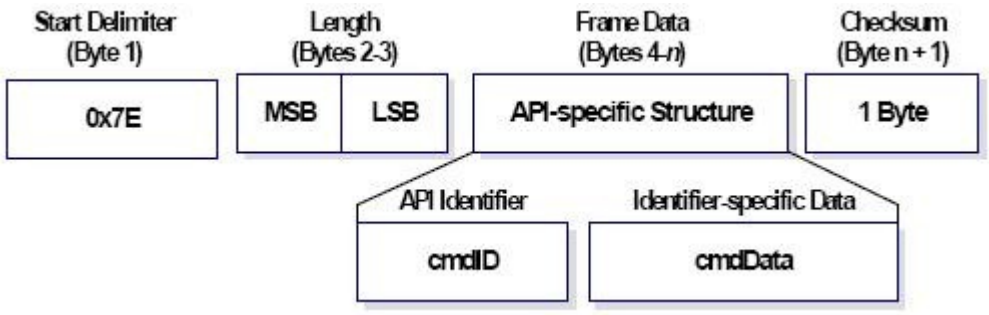


Figura 6.6 – Quadro API típico.
Fonte: Xbee manual – Zigbee protocol [8]

A tabela 6.2 a seguir ilustra os tipos de quadros existentes para a comunicação API. Os quadros “Remote Command Request” e “Remote Command Response” foram exaustivamente utilizados durante o desenvolvimento do aplicativo Java do sistema ATS.

API Identifier	Valor (Hexa)
Modem Status	8A
AT Command	08
AT Command (Ler valor do parâmetro)	09
AT Command Response	88
Remote Command Request	17
Remote Command Response	97
ZigBee transmit request	10
Explicit Addressing ZigBee Command Frame	11
ZigBee transmit status	8B
ZigBee receive packet (AO = 0)	90
ZigBee Explicit Rx Indicator (AO = 1)	91
ZigBee IO Data Sample Rx Indicator	92
Xbee Sensor Read Indicator (AO = 0)	94
Node Identification Indicator (AO = 0)	95

Tabela 6.2 – Tipo de quadros API

A seguir será exemplificada (exemplo 1) a estruturação de um comando AT via protocolo API. Nos exemplos seguintes (exemplos 2 e 3) serão demonstrados os quadros Remote Command Request e o Remote Command Resposte e suas ligações com a biblioteca utilizada para comunicação com os módulos Xbee’s.

Exemplo 1: Enviar um comando de configuração do parâmetro NJ (Permissão para novos dispositivos aderirem à rede) a um dispositivo local. Quando NJ = 0xFF o módulo sempre permite novas adesões; NJ = 0x00 – 0xFE refere-se ao número de segundos que permite novas adesões.

Quadro: 0x7E 0x00 0x05 0x08 0x01 0x4E 0x4A 0xFF 5F

0x7E – início de quadro
0x0005 – tamanho (2 bytes)

0x08 – tipo de quadro API AT command

0x01 – Identificador de quadro (Frame ID)

0x4E4A – AT command (‘NJ’)

0xFF – valor do comando

0x5F – verificador de erros (Checksum)

Ckecksum: para calcular este valor de verificação de erros no quadro usa-se a seguinte regra – Somam-se todos os bytes do quadro a ser enviado menos o delimitador de quadro e o tamanho. Os oito bits menos significativos da resposta são subtraídos de 0xFF.

Para validar um checksum somam-se todos os bytes do quadro recebido, inclusive o checksum, e a soma deverá ser igual a 0xFF.

Para o exemplo acima: [0xFF - (0x08 + 0x01 + 0x4E + 0x4A + 0xFF)] = 0x5F

Exemplo 2: Enviar um comando a um dispositivo remoto. A figura 6.7 ilustra a estrutura padrão de um quadro do tipo Remote Command Request.

Exemplo 3: Obter uma resposta a um determinado comando AT. A figura 6.8 ilustra a estrutura padrão de um quadro do tipo Remote Command Response.

A biblioteca utilizada na execução deste projeto remete a montagem e estruturação de quadros de mensagens, como os apresentados, para realizar a comunicação, iniciar uma rede, e obter dados de outros dispositivos remotos.

Os quadros remote command request foram largamente utilizados através do comando “IS”. Este comando realiza uma varredura nos pinos de saída e entrada do dispositivo e monta um quadro de resposta estruturada para indicar o estado dos pinos. Um exemplo de um pacote de resposta típico a um comando IIS é apresentado na tabela 6.3. Ao receber o pacote o programa analisa a máscara para verificar os pinos de I/O configurados como entrada e como saída e em seguida analisa os bytes seguintes bit a bit para verificar o estado de cada pino.

Os comandos “D4” e “D11” também foram largamente utilizados para poder atualizar os pinos em estado lógico alto e estado lógico baixo. Um comando desse tipo segue a estrutura apresentada na tabela 6.4.

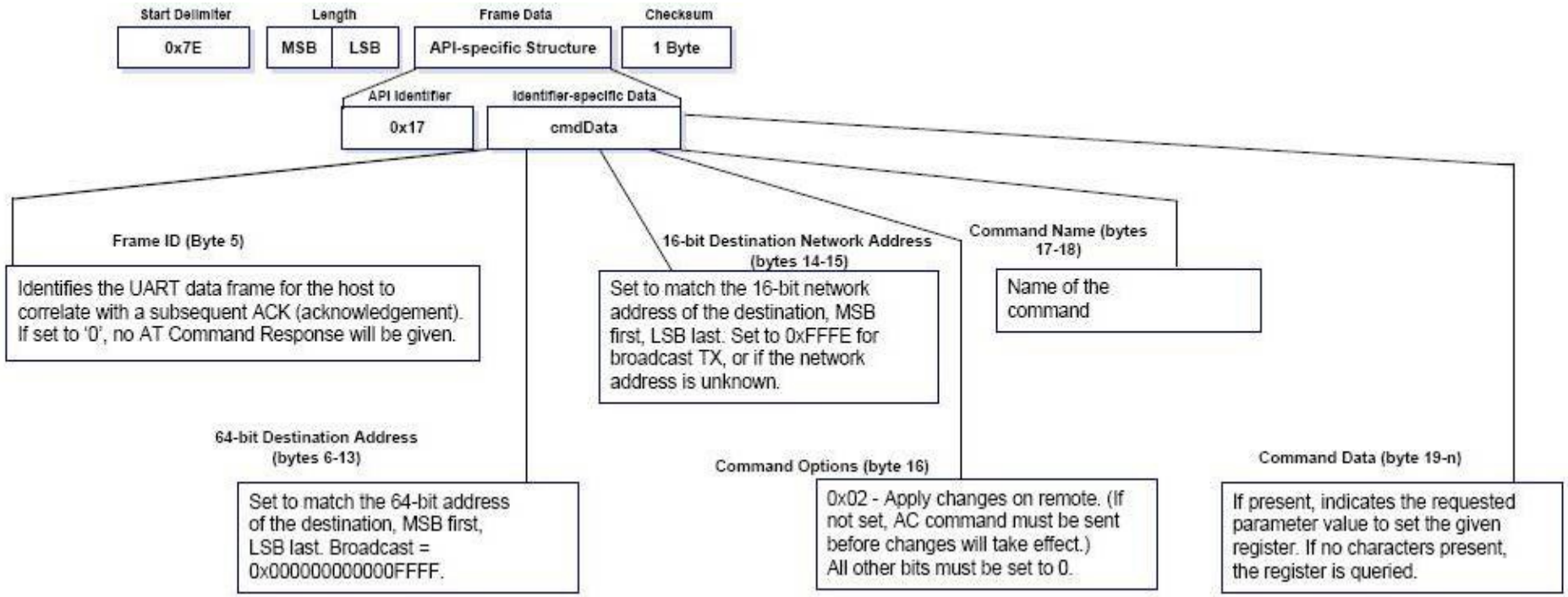


Figura 6.7 – Estrutura do quadro remote AT command Request

Fonte: Xbee manual – Zigbee protocol [8]

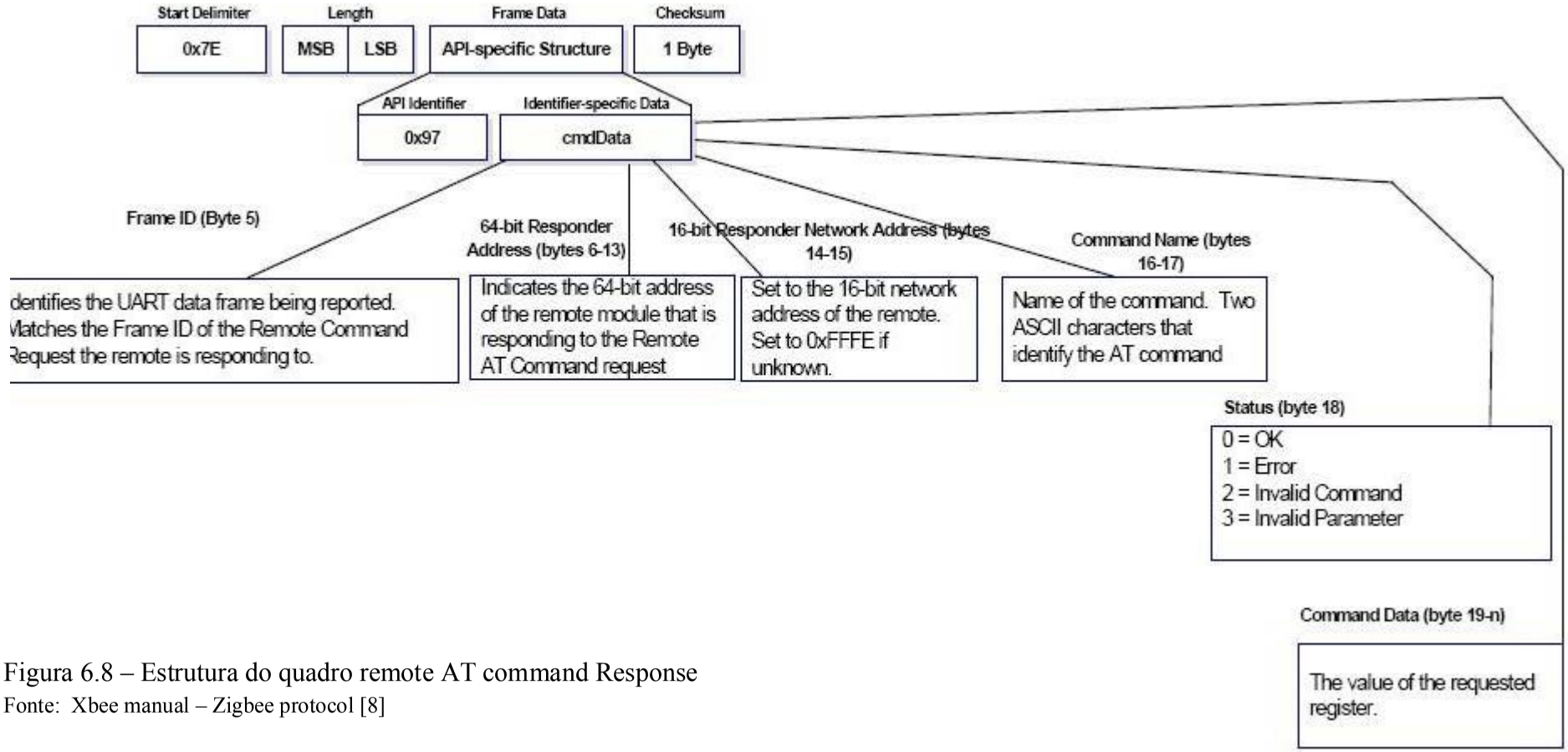


Figura 6.8 – Estrutura do quadro remote AT command Response

Fonte: Xbee manual – Zigbee protocol [8]

Bytes	Name	Description
1	Sample Sets	Number of sample sets in the packet. (Always set to 1.)
2	Digital Channel Mask	Indicates which digital IO lines have sampling enabled. Each bit corresponds to one digital IO line on the module. • bit 0 = AD0/DIO0 • bit 1 = AD1/DIO1 • bit 2 = AD2/DIO2 • bit 3 = AD3/DIO3 • bit 4 = DIO4 • bit 5 = ASSOC/DIO5 • bit 6 = RTS/DIO6 • bit 7 = CTS/GPIO7 • bit 8 = N/A • bit 9 = N/A • bit 10 = RSSI/DIO10 • bit 11 = PWM/DIO11 • bit 12 = CD/DIO12 For example, a digital channel mask of 0x002F means DIO0,1,2,3, and 5 are enabled as digital IO.
1	Analog Channel Mask	Indicates which lines have analog inputs enabled for sampling. Each bit in the analog channel mask corresponds to one analog input channel. • bit 0 = AD0/DIO0 • bit 1 = AD1/DIO1 • bit 2 = AD2/DIO2 • bit 3 = AD3/DIO3 • bit 7 = Supply Voltage
Variable	Sampled Data Set	A sample set consisting of 1 sample for each enabled ADC and/or DIO channel. If any digital IO lines are enabled, the first two bytes of the data set indicate the state of all enabled digital IO. Only digital channels that are enabled in the Digital Channel Mask bytes have any meaning in the sample set. If no digital IO are enabled on the device, these 2 bytes will be omitted. Following the digital IO data (if any), each enabled analog channel will return 2 bytes. The data starts with AIN0 and continues sequentially for each enabled analog input channel up to AIN3, and the supply voltage (if enabled) at the end.

Tabela 6.3 – Estrutura de resposta a um comando “IS”

Fonte: Xbee manual – Zigbee protocol [8]

Parâmetro	Descrição
0	Linha não monitorada
1	Reservado
2	Entrada analógica
3	Entrada digital
4	Saída digital – nível lógico baixo
5	Saída digital – nível lógico alto
6-9	Reservado

Tabela 6.4 – Parâmetros do comando D#

Capítulo 7

Funcionamento do Sistema

7.1 – Introdução

Para completar a análise do projeto, farei uma breve argumentação do que foi exposto até esse ponto e seu propósito.

A grande maioria dos CLP's apresenta uma saída modbus, para leitura dos estados e valores analógicos dos valores de campo. Para tanto o aplicativo Java, ATS, deveria ser capaz de ler diretamente via modbus o CLP. Por outro lado os estados de campo precisam ser simulados para que a validação da lógica pudesse ser aferida. Neste contexto foi escolhido o protocolo ZigBee para comunicação entre dois hardwares. Um dispositivo final fazendo o papel de chaves de campo e o outro como coordenador da rede ligado diretamente ao ATS via RS-232. Todo esse cenário e suas particularidades foram discutidos nos capítulos precedentes.

Para concluir a arquitetura proposta para o sistema de testes automáticos, falta apresentar o ATS. Durante este capítulo será discutida a lógica utilizada pelo aplicativo para se comunicar com as diversas tecnologias e ser capaz se realizar o objetivo proposto.

7.2 – Arquitetura do programa

Para alcançar o objetivo proposto o programa escrito em Java deveria ser capaz de ler e enviar dados através de suas interfaces ZigBee e Modbus. Para isto foi implementada uma classe para ambas as interfaces com o intuito de isolar a aplicação das questões técnicas e particulares de cada protocolo, focando assim na implementação do modelo de testes.

A classe XvValveOasysSimul implementa a comunicação com o CLP via modbus. O construtor da classe inicia e estabelece a comunicação podendo ser via porta serial ou via rede TCP/IP (ambos suportando modbus). Após o estabelecimento da

conexão os métodos da classe fornecem as ações necessárias modelamento dos testes a serem realizados – Tabela 7.1.

Métodos	Descrição
getValveStatus	Devolve o status da válvula em um objeto tipo XvValve
cmdOpenValve	Envia comando de abertura da válvula
cmdCloseValve	Envia comando de fechamento da válvula
cmdManutValve	Envia comando de manutenção da válvula
cmdCancelManutValve	Envia comando de cancelamento de manutenção da válvula
close	Fecha conexão modbus com o equipamento

Tabela 7.1 – Relação de métodos na classe XvValveOasysSimul

Os objetos tipo válvula contém em sua estrutura interna a condição da válvula em campo, podendo assim ser utilizado para comparação entre o valor real de campo com o padrão previsto.

A classe XvValveFieldSimul implementa a interface entre a aplicação e a rede ZigBee. Com o mesmo propósito da classe anterior, focaliza na abstração do aspecto particular do protocolo ZigBee e do módulo Xbee. A tabela 7.2 ilustra a relação de métodos utilizados para o modelamento de testes.

Métodos	Descrição
valveOpen	Simula a condição de válvula aberta
valveClose	Simula a condição de válvula fechada
valveTrans	Simula a condição de válvula em trânsito
valveFail	Simula a condição de válvula em falha
valveLocal	Simula a condição de válvula em local
valveRemote	Simula a condição de válvula em remoto
getOpenCmd	Verifica se a válvula está sobe comando de abrir
getCloseCmd	Verifica se a válvula está sobe comando de fechar
fieldStatus	Retorna um objeto XvValve com a condição de campo
Close	Fecha a conexão com a rede

Tabela 7.2 – Relação de métodos na classe XvValveFieldSimul

As duas classes descritas anteriormente pertencem a classe principal da aplicação, portanto qualquer outra classe que precisa acessar os métodos destas classes para realizar algum tipo de processamento precisa aguardar a liberação do objeto caso esteja sendo utilizado por uma outra classe.

Como cada conjunto de testes tem características diferentes e realizam verificações em partes diferentes da lógica ladder do CLP, foi implementado em threads

diferentes cada um destes conjuntos. A opção por threads também impede que a aplicação principal fique travada enquanto aguarda respostas modbus ou da própria rede ZigBee. A figura 7.1 explica como e a relação entre os objetos de interface, threads e a aplicação principal.

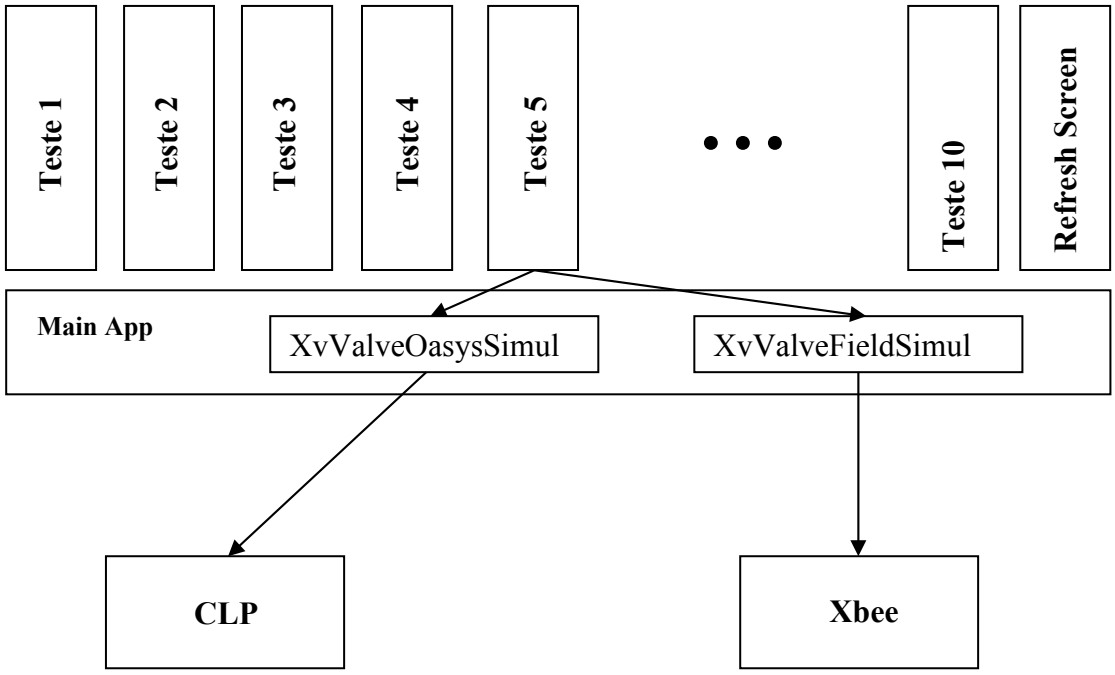


Figura 7.1 – Arquitetura do Software ATS

As setas na figura acima indicam a relação entre os objetos. O teste cinco, por exemplo, está utilizando a comunicação entre os dispositivos, enquanto a atualização de tela (Refresh Screen) deve aguardar o teste 5 liberar o objeto para poder atualizar a tela.

Esta estrutura se torna eficiente também pelo ponto de vista da versatilidade. Novos testes de outros equipamentos, ou outras lógicas podem ser implementas simplesmente criando uma nova thread que contenha o cabeçalho das threads já existentes, já que a Main App e as interfaces são as mesmas para todas.

7.3 – Interface gráfica

Para o acompanhamento dos testes foi elaborada um interface gráfica de fácil compreensão e diagnóstico dos eventos ocorridos durante um determinado teste. Um modo manual de operação foi implementado para realizar checagens manuais e visuais no acompanhamento da lógica no CLP e para “debug” do programa durante o seu desenvolvimento. O modo automático é a implementação do objetivo proposto pelo

ATS, onde cada teste é realizado em seqüência, podendo ser acompanhado pelas mensagens exibidas no quadro 'Test Process', conforme figura 7.2.

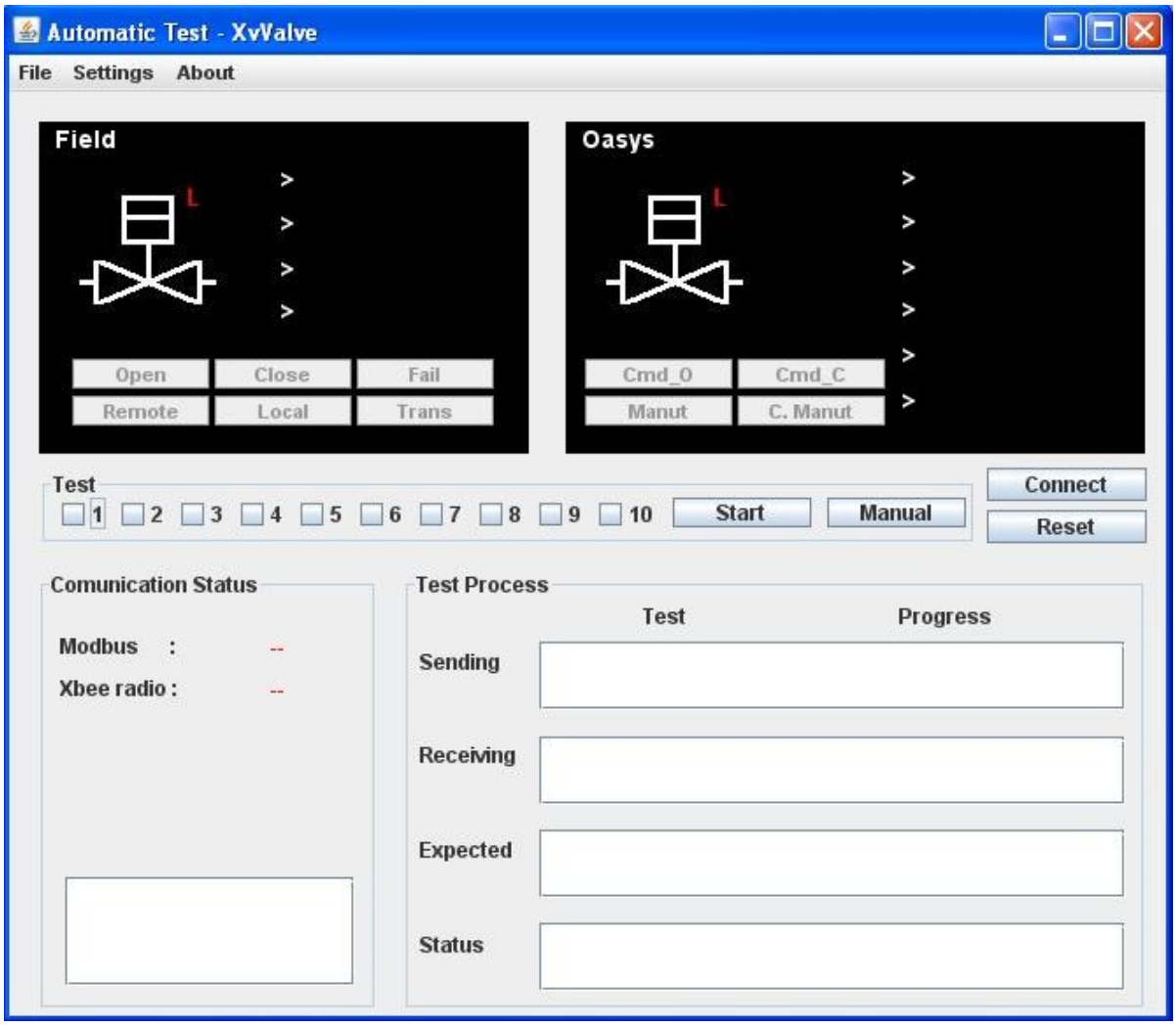


Figura 7.2 – Interface do programa ATS

7.4 – Testes programados

Em cada thread foi modelado um tipo exclusivo de teste visando atestar o correto funcionamento do lógico do CLP – ladder da figura 3.4. A seguir será apresentado cada um dos dez testes modelados dentro do sistema ATS.

Teste 1: Realizado para obter a consistência das informações vindas de campo. Um exemplo de teste é simular via Xbee a válvula aberta e verificar se a lógica sinaliza corretamente a válvula aberta. A tabela 7.3 ilustra a sequência testada.

Simulado - Xbee	Retorno esperado - modbus
Aberta	Aberta
Fechada	Fechada
Trânsito	Trânsito
Falha	Falha
Local	Local
Remoto	Remoto

Tabela 7.3 – Sequência do teste 1

Teste 2: Realizado para verificar se os sinais de comando estão chegando no campo. No momento em que o sistema ATS verifica que o comando chegou em campo ele simula o estado referente ao comando e afere se o retorno está como o esperado. A tabela 7.4 ilustra a sequência do teste 2.

Comando - modbus	Simulado - Xbee	Retorno esperado - modbus
Abrir	Aberta	Aberta
Fechar	Fechada	Fechada
Manutenção	-	Manutenção
Cancela manutenção	-	Estado de campo (aberta ou fechada)

Tabela 7.4 – Sequência do teste 2

Teste 3: Verifica o teste do time-out de abrir. Este estado refere-se à impossibilidade de abrir a válvula perante um comando de abrir. A sequência deste teste é apresentada na tabela 7.5.

Comando - modbus	Simulado - Xbee	Retorno esperado - modbus
Abrir	Fechada	Time-out Abrir (depois de 15s)
Abrir	Fechada	Fechada (retira o time-out)
-	Aberta	Aberta
Abrir	Trânsito	Time-out Abrir (depois de 15s)
Abrir	Trânsito	Trânsito (retira o time-out)
-	Aberta	Aberta

Figura 7.5 – Sequência do teste 3

Teste 4: Verifica o teste do time-out de fechar. Este estado refere-se à impossibilidade de abrir a válvula perante um comando de fechar. A sequência deste teste é apresentada na tabela 7.6.

Comando - modbus	Simulado - Xbee	Retorno esperado – modbus
Fechar	Aberta	Time-out fechar (depois de 15s)
Fechar	Aberta	Aberta (retira o time-out)
-	Fechada	Fechada
Fechar	Trânsito	Time-out Fechar (depois de 15s)
Fechar	Trânsito	Trânsito (retira o time-out)
-	Fechada	Fechada

Figura 7.6 – Sequência do teste 4

Teste 5: Verifica o teste do time-out de trânsito. Este estado refere-se à permanência da válvula no estado não fechado e não aberta por um determinado período. A sequência deste teste é apresentada na tabela 7.7.

Comando - modbus	Simulado - Xbee	Retorno esperado – modbus
-	Trânsito	Time-out trânsito (depois de 15s)
-	Aberta	Aberta (retira o time-out)
-	Trânsito	Time-out trânsito (depois de 15s)
-	Fechada	Fechada (retira o time-out)
-	Trânsito	Time-out trânsito (depois de 15s)
Abrir	Trânsito	Trânsito (retira o time-out)
-	Trânsito	Time-out trânsito (depois de 15s)
Fechar	Trânsito	Trânsito (retira o time-out)

Figura 7.7 – Sequência do teste 5

Teste 6: Verifica a lógica de operação local/remoto. Quando a válvula está em local, sua operação é local, logo deve ignorar comandos remotos. Em modo remoto deve aceitar os comandos remotos. A sequência utilizada é apresentada na tabela 7.8

Comando - modbus	Simulado - Xbee	Retorno esperado – modbus
-	Local	Local

Abrir	-	(não pode chegar o comando)
Fechar	-	(não pode chegar o comando)

Figura 7.8 – Sequência do teste 6

Teste 7: Analisa a lógica de manutenção do instrumento. Utilizada para evitar leituras erradas de um equipamento em falha ou com algum problema de comunicação. A sequência utilizada é apresentada a seguir na tabela 7.9.

Comando - modbus	Simulado - Xbee	Retorno esperado – modbus
Manutenção	-	Manutenção
Abrir	-	(não pode chegar o comando)
Fechar	-	(não pode chegar o comando)
Cancelar Manutenção	-	Estado de campo

Figura 7.9 – Sequência do teste 7

Teste 8: Analisa a lógica de falha. Este estado ocorre quando os dois contatos de campo aberta e fechada estão atuados. Neste contexto a válvula deve ficar incapacidade de receber comandos. A sequência utilizada para modelamento deste teste esta apresentada na tabela 7.10.

Comando - modbus	Simulado - Xbee	Retorno esperado – modbus
-	Falha	Falha
Abrir	-	(não pode chegar o comando)
Fechar	-	(não pode chegar o comando)

Figura 7.10 – Sequência do teste 8

Teste 9: Verifica a sequência de comandos alternados. Quando a válvula recebe um comando de abrir e logo após um de fechar o último comando dado deve prevalecer. Nunca os dois comando devem estar atuando ao mesmo tempo em cima da válvula. A sequência de verificação desta situação é ilustrada na tabela 7.11.

Comando - modbus	Simulado - Xbee	Retorno esperado – modbus
-	Trânsito	Trânsito
Abrir	Trânsito	(comando chegou)
Fechar	Trânsito	(comando chegou)
Abrir	Trânsito	(comando chegou)
-	Aberta	Aberta

Figura 7.11 – Sequência do teste 9

Teste 10: Verifica se o comando está aplicado à válvula mesmo ela já estando na condição desejada. Se a válvula já está aberta o comando de abrir não deve chegar no campo. A sequência deste teste é apresentada a seguir – Figura 7.12

Comando - modbus	Simulado - Xbee	Retorno esperado – modbus
-	Aberta	Aberta
Fechar	Aberta	(comando chegou)
Abrir	Aberta	(comando não pode chegar)

Figura 7.11 – Sequência do teste 12

Capítulo 8

Conclusão

Os principais resultados obtidos na realização deste projeto foi a padronização dos procedimentos de testes. Uma vez que qualquer modelo pode ser programado no ATS podemos com este sistema melhorar a qualidade dos testes e obter maior rapidez na realização dos mesmos.

O sistema ainda poderá ser modificado, visto que toda a programação foi modularizada do ponto de vista da comunicação com a rede ZigBee e a linha modbus. As classes utilizadas para realizar esta comunicação podem ser implementadas sem nenhuma modificação para o modelamento de novos testes, podendo assim abranger uma vasta gama de equipamentos e de diferentes lógicas e padrões.

Bibliografia

[1] _____. "*SCADAPack Light controller user and reference manual*" – Control Microsystems, Kanata, Ontario, Canada. http://www.scadapack.com.br/Download.html. (Acesso em 20 agosto 2009).

[2] _____. "*Curso de Controladores lógicos programáveis*" – UERJ/LEE, Rio de Janeiro, RJ, Brasil. http://www.lee.eng.uerj.br/downloads/cursos/clp/clp.pdf. (Acesso em 20 agosto 2009).

[3] _____. "*Modbus application protocol specification*" – ModBus-IDA, Hopkinton, Massachusetts, EUA. http://www.modbus.org/docs/Modbus_ Application_Protocol_V1_1b.pdf. (Acesso em 17 março 2009).

[4] FILHO, C. S., "*Protocolos Orientados a Caractere*" – UFMG, Belo Horizonte, Pampulha, Brasil. http://www.scribd.com/doc/16180270/ProtocolosCaracterMBUS. (Acesso em 1 setembro de 2009).

[5] CAPELLI, A., "*Automação Industrial - Controle do Movimento e Processos Contínuos"*. São Paulo, Editora Érica, 2007. p.23.

[6] _____. "*Latest ZigBee specification including the pro feature set***"** – ZigBee Alliance. http://www.zigbee.org/Products/TechnicalDocumentsDownload/tabid/ 237/Default.aspx. (Acesso em 17 março 2009).

[7] _____. "*Jamod API Documentation*" – Jamod, http://jamod.sourceforge.net/api/index.html. (Acesso em 5 maio 2009).

[8] _____. "*Product Manual v1.x.4x - ZigBee Protocol***"** – Digi International Inc. ftp1.digi.com/support/documentation/90000866_C.pdf. (Acesso em 18 março 2009).

[9] ZigBee Alliance. “*ZigBee Specification*” – ZigBee Alliance, http://www.zigbee.org. (Acesso em 25 março 2009).