UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
ENGENHARIA DE COMPUTAÇÃO

ANTÔNIO AUGUSTO DA FONTOURA

Plataforma de Comunicação e Controle para Integração de Nós Estáticos e Móveis em Redes de Sensores Sem Fio

Trabalho de Conclusão apresentado como
requisito parcial para a obtenção do grau de
Engenheiro de Computação

Prof.Dr. Carlos Eduardo Pereira
Orientador

Porto Alegre, Junho de 2012

CIP – CATALOGAÇÃO NA PUBLICAÇÃO

da Fontoura, Antônio Augusto

Plataforma de Comunicação e Controle para Integração de Nós Estáticos e Móveis em Redes de Sensores Sem Fio / Antônio Augusto da Fontoura. – Porto Alegre: UFRGS, 2012.

57 f.: il.

Trabalho de Conclusão – Universidade Federal do Rio Grande do Sul. Engenharia de Computação, Porto Alegre, BR–RS, 2012. Orientador: Carlos Eduardo Pereira.

1. Estação de Solo. 2. Controle de Missões. 3. Rede de Sensores Sem Fio. 4. Rede de Sensores Heterogênea. 5. Veículo aéreo não tripulado. 6. VANT. 7. XBee. I. Pereira, Carlos Eduardo. II. Título.

“If I have seen farther than others,
it is because I stood on the shoulders of giants.”
— Sir Isaac Newton

AGRADECIMENTOS

Primeiramente eu gostaria de agradecer à minha família, que sempre esteve junto comigo em todos os momentos e me apoiou nas decisões mais difíceis, mesmo sabendo que a distância nos separaria por longos períodos.

Aos meus colegas de faculdade que se tornaram grandes amigos. Sem a parceria para as rodadas de truco no intervalos de física, para as tardes de estudo, para as noites viradas por trabalhos de última hora, talvez nao teria chegado ao fim dessa jornada.

Aos velhos amigos de infância e ao novos de Alemanha. Que estiverem presentes nas alegrias mas também nas horas difíceis.

Aos colegas de trabalho pelas inúmeras oportunidades de aprendizagem.

Gostaria de agradecer também à Universidade Federal do Rio Grande do Sul que, junto com o Instituto de Informática, ofereceu uma das melhores graduações de engenharia de computação do Brasil. Ao meu orientador por conseguir toda a infraestrutura necessária para o desenvolvimento desse trabalho. Agradeço ao Edison Pignaton , Ivan Muller, Rodrigo Allgayer, e Wiliam Silva por auxliarem diretamente para a consolidação do trabalho, tanto com ideias como em ajudas técnicas.

SUMÁRIO

LISTA DE ABREVIATURAS E SIGLAS

API	Application Programming Interface
ASCII	American Standard Code for Information Interchange
CRC	Cyclic Redundancy Check
DC	Direct Current
EMF	Electromotive Force
FFD	Full Function Device
HR	High Rate
IEEE	Institute of Electrical and Electronics Engineers
ISO	International Organization for Standards
LLC	Logical Link Control
LR	Low Rate
LSB	Least Significant Bits
MAC	Media Access Control
MSB	Most Significant Bits
OEM	Original Equipment Manufacturer
OSI	Open System Interconnect
PAN	Personal Area Network
PDA	Personal Digital Assistant
PWM	Pulse Width Modulation
QoS	Quality of Service
RFD	Reduced Function Device
RSSF	Rede de Sensores sem Fio
UAV	Unmanned Aircraft Vehicle
VANT	Veículo Aéreo Não Tripulado
XML	Extensible Markup Language
WPAN	Wireless Personal Area Network
WSN	Wireless Sensors Network

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE LISTAGENS

RESUMO

Os avanços tecnológicos relacionadas a diferentes tipos de sensoriamentos e rádio-frequência, diminuindo os custos e consumo de energia, permitiram que Redes de Sensores Sem Fio (RSSF) ganhassem foco da comunidade científica, e se mostraram de grande aplicabilidade. Além disso, o uso conjunto de sensores de diferentes habilidades, nodos móveis e estáticos, formando uma rede heterogênea, mostra-se muito mais abrangente em diferentes aplicações. O uso de veículos aéreos não tripulados (VANTs) como um nó da rede a torna extremamente versátil e com muito mais recursos. Nesse âmbito, este trabalho propõe a centralização de informações dos diferentes tipos de sensores da rede, criando interfaces compatíveis com os nodos disponíveis. Com isso, foi possível a implementação de um controle da rede com o objetivo do cumprimento de uma missão previamente descrita.

Palavras-chave: Estação de Solo, Controle de Missões, Rede de Sensores Sem Fio, Rede de Sensores Heterogênea, Veículo aéreo não tripulado, VANT, XBee.

ABSTRACT

Technological advances related to different types of sensors and radio-frequency, reducing costs and energy consumption have enabled the Wireless Sensor Networks to gain focus in the scientific community, and have shown their huge potential in differents kinds of applications. Furthermore, the use of different sensor skills combined, like static and mobile nodes, creating a heterogeneous network, also has become to be a lot more embracing. The use of unmanned aircraft vehicle (UAVs) as a network node makes it extremely versatile, with more features. In this context, this work proposes to centralize the information of different types of sensors in a network, creating compatible interfaces with the available nodes. Thus, it is also possible to control the network in order to accomplish a mission previously specified.

Keywords: WSN, UAV, Heterogeneous WSN, Ground station, Base station, Mission control.

1 INTRODUÇÃO

Nos últimos anos o avanço de diversas tecnologias relacionadas a veículos aéreos não tripulados (VANTs) e rede de sensores sem fio (RSSF) fizeram que a sua integração se tornasse possível, e de extremo potencial.

Surgiram padrões e especificações para redes de comunicação sem fio para aplicações de baixa taxa de transferência e baixo alcance de sinal, e por consequência a redução drástica de consumo de energia e custos de produção e manutenção dos equipamentos relacionados. Por isso, as RSSF passaram a ser estudadas e aplicadas em diferentes aéreas da engenharia como, por exemplo, no monitoramento estrutural de pontes [22] e em sistemas de irrigações agrícolas [7].

A inserção de diferentestipos de sensores em uma mesma rede permite o enriquecimento das informações providas, aumentando também a sua precisão em relação ao ambiente monitorado. Da mesma forma, sensores estáticos com informações básicas, podem ser complementados com sensores móveis mais sofisticados.

Em paralelo, os VANTs trouxeram uma vasta gama de recursos, que, apesar de complexos e por vezes mais caros que os nodos estáticos, criam diversas possibilidades para aumentar a qualidade e precisão no que diz respeito ao monitoramento de regiões específicas. Pela capacidade desses veículos de se moverem em três dimensões, de terem a possibilidade de embarcarem sensores mais sofisticados e de serem autônomos em sua mobilidade, podem ser de grande utilidade como nós móveis de uma RSSF.

Porém a centralização das informações de diversos dispositivos distintos com diferentes complexidades se torna mais custosa.

1.1 Motivação

As redes de sensores sem-fio heterogêneas trazem um grande potencial para diversas aplicações. Em [20], é evidenciado o uso da cooperação de veículos aéreos não tripulados com sensores estáticos em solo na agricultura de precisão, monitoramento de linhas de transmissão e apoio à segurança pública. Além disso, mostra o interesse de empresas no desenvolvimento da rede de sensores para essas aplicações.

A possibilidade do estudo e desenvolvimento de RSSF muito mais complexas e capacitadas é apresentado em [8]. O emprego da cooperação entre os nodos da rede, tanto estáticos como dinâmicos, apresenta um aumento da performance em missões de vigilância.

1.2 Trabalho Proposto

O trabalho consiste no desenvolvimento de um software que centraliza informações e gerencia uma RSSF. A estação de solo, como é chamada, deve montar um ambiente a partir de um arquivo de entrada contendo a descrição de alto nível de uma missão específica.

O objetivo é montar uma rede com topologia estrela usando módulos XBee Pro S1 (IEEE 802.15.4), e através de suas respectivas placas de desenvolvimento, configurá-los para que sejam sensores estáticos partícipes de uma RSSF. Desta rede também farão parte Veículos Aéreos Não-Tripulados (VANTs), e para isto implementa-se neste trabalho uma interface de comunicação com a plataforma quadricóptero desenvolvido pela empresa *Skydrones* [21], que representa o nó sensor móvel da rede. Desta forma, todas as informações disponíveis pelo software embarcado do VANT podem ser mostradas ao usuário, bem como quadricóptero pode interagir com nós estáticos da rede, recebendo assim comandos não só da estação de solo, mas também através de eventos gerados pelos nós estáticos.

Com isso, tarefas de alta complexidade de gerenciamento, como a vigilância de regiões críticas, podem ser feitas de forma muito mais simplificada por parte do operador. A figura 1.1 mostra um diagrama da estrutura compreendida pelo trabalho.

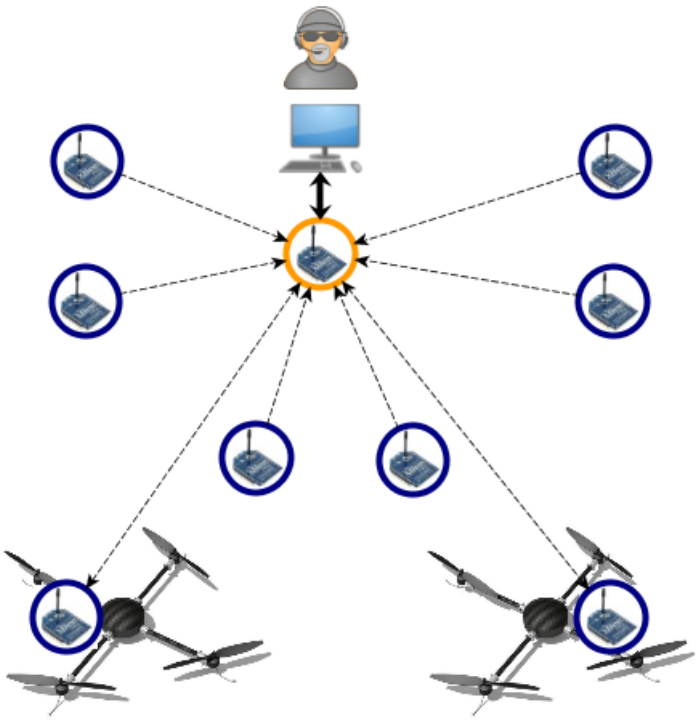

Figura 1.1: Estrutura da rede sem-fio proposta

O software deve possuir também uma arquitetura modular para que diferentes interfaces de variados periféricos possam ser facilmente adaptados e incorporados ao sistema.

2 REDE DE SENSORES SEM-FIO

Uma RSSF consiste na composição de um ou mais nós sensores interconectados sem a presença de cabos para realizar trocas de informações. Tem por objetivo monitorar e até controlar o ambiente em que está inserida.

As pesquisas em torno de diferentes redes de comunicação assumiam na sua maioria um sistema cabeado, com ilimitada energia e por isso sem economia de poder computacional e com alta taxa de transferência de dados sempre que necessário [2]. Contudo as redes de sensores sem-fio apresentaram uma série de restrições que deveriam ser levadas em conta em seu desenvolvimento. Os avanços na tecnologia de rádiofrequência, de microprocessadores e de sensoriamento, através das tecnologias de micro sistemas eletromecânicos, permitiram que tais restrições pudessem ser em parte solucionadas e consequentemente que as RSSF estivessem cada vez mais presentes nas pesquisas científicas e nas mais diversas aplicações.

De modo geral, as redes de sensores sem-fio possuem numerosos nós estáticos com limitações de custos e consumo de energia. Entretanto, necessitam de um poder computacional suficiente para lidar com trocas de mensagens e processar dados capturados de seus sensores. Alem disso faz-se necessária a presença de receptores e transmissores de rádiofrequência com potência necessária para a comunicação com os nós vizinhos. O tamanho reduzido, aliado a essas limitações, acabam sendo de grande valia quando aplicados em regiões críticas e remotas ou também em ambientes hostis.

Com essas características, as RSSFs podem cobrir uma região geográfica considerável apenas sendo replicados em um número suficientemente grande de nós. Entretanto, por possuírem apenas sensores básicos para captura de fenômenos a sua volta como, por exemplo, sensores de temperatura e de presença, sua capacidade individual de gerar informações relevantes é limitada. O emprego de um número maior desses dispositivos distintos permite que a rede, de forma coordenada, possa realizar a fusão de dados provenientes de seus nós e enriquecer a informação gerada pela rede [3]. Por esse motivo, a heterogeneidade da rede, ou seja, o uso de nós com diferentes habilidades tais como tipos distintos de sensores e a cobertura da comunicação e de captação de fenômenos externos, podem enriquecer de forma significativa as informações geradas pelo sistema como um todo. Otimizações de custo da rede através do correto posicionamento dos diferentes tipos de sensores em uma determinada região de monitoramento é discutido em [6], assim como a análise referente ao aumento da eficiência energética da rede de sensores como um todo é analisado em [14].

Os sistemas de RSSF costumam possuir uma estação base, de maior poder computacional, que coleta as informações dos nós da rede e eventualmente toma decisões referentes a determinados eventos [13]. A figura 2.1 representa uma aplicação de uma rede com uma estação base presente. No exemplo, é recebida a informação que um determinado

número de sensores, por exemplo, detectou a presença de fogo em certa região de floresta e, a partir disso, aciona os alarmes.

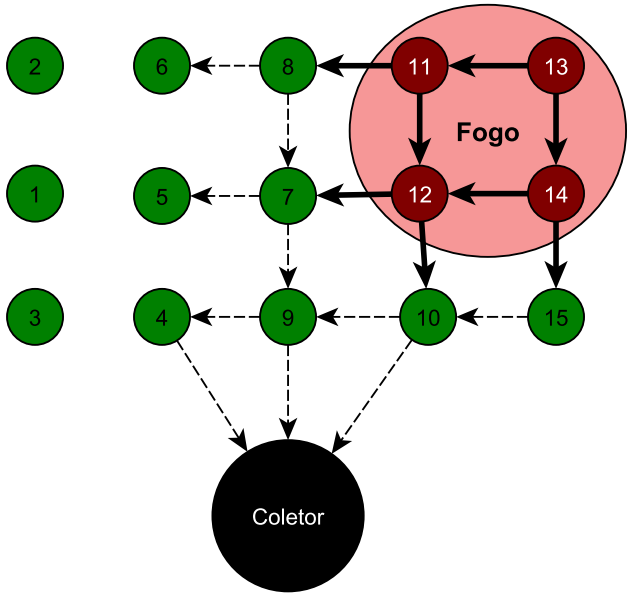

Figura 2.1: Exemplo de emprego de nós estáticos e um estação base

O uso de sensores mais sofisticados como, por exemplo, radares e câmeras infravermelho, não podem ser descartados em aplicações como as de vigilância. Por terem um alto custo, acabam sendo empregados em menor número, e consequentemente, abrangendo uma área de captura muito menor. Entretanto, as informações capturadas se tornam mais ricas, trazendo ao sistema um detalhamento maior do ambiente. A instalação de uma câmera de vigilância em um avião, por exemplo, torna a sua abrangência geográfica praticamente ilimitada (desconsiderando limitações de energia). Esse recurso torna possível o monitoramento muito mais próximo de pontos de interesse, alem de cobrir regiões de forma muito mais granular. No entanto tal solução não é auto-suficiente em uma missão de monitoramento visto que o nó móvel não é onipresente na região de vigilância, podendo perder eventos cruciais em áreas não cobertas em determinado momento.

Uma forma de se implementar nós móveis é através da instalação de sensores em Veículos Aéreos Não-Tripulados (VANTs). Esta alternativa confere uma grande flexibilidade para a rede, uma vez que permite a movimentação do sensor em 3 dimensões. Com isto além do menor custo, tem-se a vantagem de poder usá-lo em ambientes hostis, onde a utilização de aeronaves tripuladas seria inviável.

Com isso em mente, o uso conjunto de nós estático e dinâmicos mostra-se extremamente benéfico, podendo aumentar o grau de relevância e precisão da rede. A simplicidade de determinados sensores pode auxiliar e complementar a ação de nós sofisticados. Na situação apresentada anteriormente em que uma câmera é instalada em um avião, pode-se adicionar o monitoramente realizado por dezenas ou até centenas de sensores de presenças espalhados pela região de interesse. Assim que um no estático detecta algum movimento um alarme pode ser emitido ao avião de forma com que esse replaneje a sua trajetória, indo em direção ao ponto em que o evento ocorreu.

As RSSF podem ser empregadas nas mais variadas aplicações. Em [7], uma rede de sensores é usada para auxiliar na irrigação de plantações na Tunisia. Utiliza o recurso de

uma estação base para realizar a centralização das informações provenientes dos sensores de umidade instalados na região de interesse e gerar relatórios, para que, a partir disso, o gerenciamento do sistema possa ser feito.

O desenvolvimento do monitoramento de bancos de baterias é proposto em [18]. A rede de sensores proposta visa garantir que baterias auxiliares de *nobreaks* estejam funcionando no momento em que serão exigidas, ou seja, quando a fonte de energia principal do sistema apresentar defeito.

Em função da decorrência constante de terremotos no Japão, uma rede de sensores sem-fio para o monitoramento estrutural de um ponto é apresentado em [22]. Sensores de vibração são instalados por todo o comprimento da ponte em questão e são enviados uma estação base a partir de nós roteadores.

3 TECNOLOGIAS DE COMUNICAÇÃO SEM FIO

3.1 Introdução

Durante a segunda metade do século passado apresentou-se um crescimento exponencial do uso de redes de comunicação sem-fio, e na medida em que as tecnologias foram se desenvolvendo, novas especificações foram surgindo para determinados conjuntos de aplicações como redes de telefonia móvel, por exemplo. Na metade da década de 80 percebeu-se a necessidade de definições com relação ao uso de bandas de frequência sem a necessidade de licenças governamentais. As redes locais sem-fio (WLANs) foram então criadas e uma série de características foram especificada como, por exmplo, taxas de transferência de 11 Mbps, alcances médios de ate 100 metros e tratamentos de erros. Entretanto, redes locais ainda mais restritas, com menor complexidade e consequentemente com menor custo e menor consumo de energia passaram atrair muito interesse da comunidade científica.

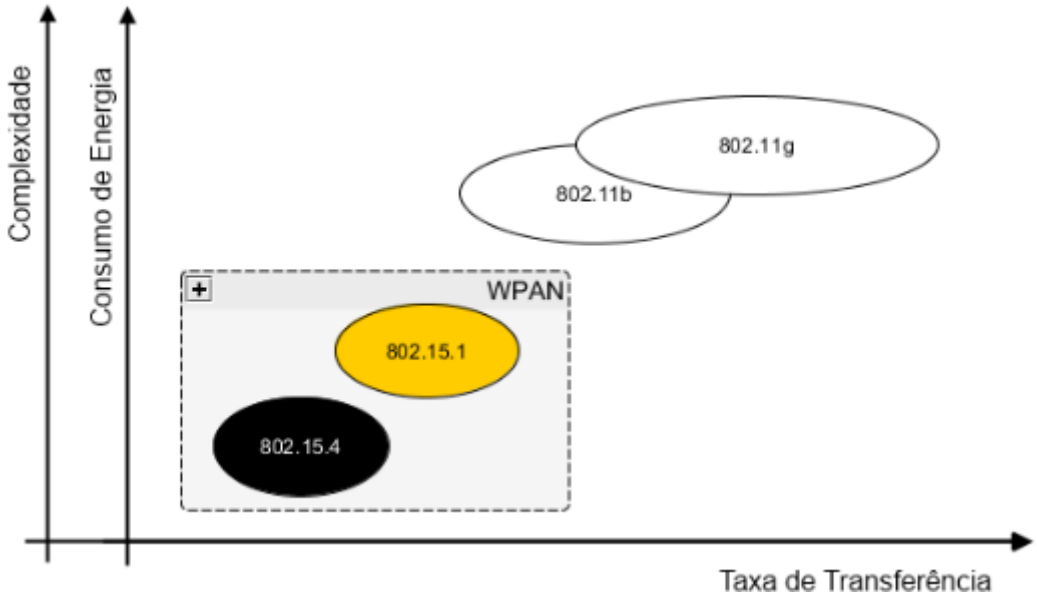

Figura 3.1: Espaço operacional entre WPANs e WLANs

A IEEE 802.15 foi criada para cobrir esse propósito [12], de especificar redes pessoais sem-fio (WPANs) e foi dividida em sete grupos de trabalho. Dessas, três grupos receberam um foco maior por parte dos comitês, a IEEE 802.15.3, High Rate WPAN (HR-WPAN) voltada a WPANs com mais alta taxa de transferência destinadas à aplicações multimídia que necessitam de alta qualidade de serviço (QoS), IEEE 802.15.4 Low

Rate WPAN (LR-WPAN) para baixas taxas de transferência destinado a aplicações industriais, residenciais e médicas, e a IEEE 802.15.1 voltada para comunicação local de Celulares e PDAs.

Dois desses padrões passaram a atrair a atenção de empresas mundialmente reconhecidas, dispostas a desenvolver referências para comunicações de baixo custo e menor alcance. O 802.15.1 juntamente como o Bluetooth Special Interest Group (SIG) formado por empresas como a Ericsson e Nokia, e o 802.5.4 em conjunto com a ZigBee Alliance formada por empresas como a Motorola e Phillips. Em primeira análise, ambas apresentam algumas similaridades, porém uma série de características destinadas a aplicações específicas os diferenciam[4].

3.2 ZigBee x Bluetooth

O Bluetooth foca principalmente no alcance curto de sinal em dispositivos móveis enquanto o ZigBee enfatiza baixo consumo de energia, interconexão e padrão aberto de comunicação. Mesmo que o padrão 802.15.4 defina a cobertura de sinal na mesma ordem que o 802.15.1, o ZigBee possui um potencial muito maior para longo alcance. Alem disso, o Bluetooth foi desenvolvido para que dispositivos permaneçam ativos a maior parte do tempo, prontos para iniciar qualquer conexão ou iniciar uma transferência de dados. ZigBee adere à técnica de hibernação para a economia de energia, porém foi concebido de forma a retornar ao modo ativo quando necessário de forma muito rápida.

A versatilidade do ZigBee se destaca frente ao Bluetooth. Além da comumente usada banda de frequência de 2.4 GHz, o padrão ainda define faixas na ordem de 868 MHz e 915 MHz (dependendo da localização geográfica), e em revisões mais recentes da especificação [1] adiciona as faixas de frequências de 6289.6 MHz a 9185.6 MHz e de 433MHz. Teoricamente permite em sua rede um conjunto de até 65 mil dispositivos, além de manter um padrão de comunicação aberto para que qualquer interface possa ser conectada. Por outro lado, o Bluetooth especifica seu meio físico apenas para a faixa de 2.4 GHz e possui a limitação de apenas 8 dispositivos conectados a sua rede em uma configuraçao simples. Adiciona também, protocolos específicos para a interoperabilidade entre equipamentos, o que torna mais complexa a conexão e transmissão de dados.

característica	**Wi-Fi**	**Bluetooth**	**ZigBee**
Frequencia	2.4GHz/5.3GHz(1)	2.4GHz	433 MHz (2), 868 MHz, 902-928 MHz, 2.4 GHz, 6289.6-9185.6MHz (2)
Taxa de Transferência	250 Mbps	3 Mbps	250 Kbps
Custo/Complexidade (3)	> 6	1	0.2
Autonomia (Energia)	Horas	Dias	Anos
Alcance do sinal	250m	10m a 100m	10m a 400m
Número de dispositivos na rede	255	8	65,000

Notes: (1) 802.11n, (2) 802.15.4f, (3) em [12].

Tabela 3.1: Comparativo entre padrões wireless IEEE

Um exemplo claro de aplicação para o padrão ZigBee podem ser pequenos sensor de temperatura instalados em vários cômodos de uma casa, todos eles com dispositivos de transmissão de dados sem-fio segundo as especificações. Esses sensores necessitam

reportar as temperaturas medidas apenas algumas vezes por hora, limitando ao máximo o consumo de energia e taxa de transferência. É facilmente perceptível que o exemplo dado se aplica diretamente a um nó de uma RSSF. Em contrapartida, tal situação não teria a mesma eficiência caso a tecnologia Bluetooth fosse usada. Os sensores ficariam ativos constantemente para evitar a alta latência de criar uma nova conexão e dessa forma o tempo de vida de suas baterias se reduziria drasticamente, tornando a manutenção do sistema extremamente custosa e trabalhosa.

A tabela 3.1 apresenta um comparativos entre redes WLAN, WPAN e LR-WPAN, evidenciando o foco de cada padrão proposto.

3.3 Protocolo IEEE 802.15.4

A IEEE 802.15.4 tem por objetivo especificar as duas primeiras camadas do padrão ISO-OSI, camada física e camada de enlace, mais especificamente a camada MAC. Em contrapartida, a ZigBee Alliance é responsável pela definição das camadas de rede, transporte e aplicação.

3.3.1 Camada Física

A camada física é ponto crucial para um dos principais objetivos do padrão, que é o baixo consumo de energia. Por isso a IEEE 802.15.4 a definiu levando em consideração questões técnicas, como o tipo de modulação de sinal para uma maior eficiência energética, por exemplo, como também questões não técnicas como alguns requisitos regidos com regulamentações governamentais [12].

O padrão possui a característica de ser multi-banda, ou seja, disponibilizar mais de uma faixa de frequência para o uso. Inicialmente foram especificadas as bandas de 2.4 GHz e 868/915 MHz, e mais recentemente 433 Mhz e 6289.6 MHz a 9185.6 MHz.

O amplo uso por outras especificações e dispositivos torna a faixa de 2.4GHz interessante no ponto de vista de custos para os fabricantes, já que o mercado e consequentemente a produção é muito maior. Por outro lado interferências causadas pelo congestionado meio dependendo da situação, podem trazer prejuízos à rede. Por esse motivo, outras faixas de frequências foram apresentadas como opções para os projetistas, cabendo a eles a análise dos requisitos e do meio em que rede será instalada para evitar possíveis ruídos em sua rede. Entretanto, ha penalizações referentes ao uso de frequências mais baixas. Apesar de um alcance maior de sinal, a taxa de transferência cai drasticamente, de 250Kbps para 20Kbps.

Técnicas de modulação de sinal específicas e desenvolvidas para aumentar a eficiência energética também foram estudadas e definidas para atender às restrições propostas inicialmente [12]. A modulaçao é baseada no método de espalhamento espectral (DSSS) o qual resulta num custo menor de implementação dos circuitos digitais.

3.3.2 Camada de Enlace

A camada de enlace é subdividida em duas outras camadas, LLC e MAC. A LLC segue a especificação da 802.2 assim como tanto outros padrões como, por exemplo, 802.3 e 802.11. Por outro lado, a subcamada MAC é mais próxima do hardware e consequentemete mais dependente da implementação da camada física. Por possuir apenas uma parcela de primitivas de serviço em relação à 802.15.1 (Bluteooth), a camada MAC é bem menos complexa, sendo facilmente aplicável em dispositivos mais limitados [5].

Implementa recursos de associação e dissociação, pacotes ACK, pacotes de anúncio

(*beacon*) e validação de pacotes [5]. Além disso, permite a implementação de topologias múltiplas e com menor complexidade.

3.3.3 Topologias

A IEEE 802.15.4 define dois tipos de dispositivos que podem integrar uma rede. Os dispositivos com funções completas (FFD) e os dispositivos com funções limitadas (RFD).

FFDs podem assumir o papel de simples dispositivos, de coordenador ou de coordenador de rede pessoal (PAN). Já os RFDs tem por finalidade o emprego em aplicações extremamente simples, com baixa taxa de transferência de dados. Desse modo RFDs conversam apenas com FFDs.

Com os integrantes da rede definidos, uma LR-WLAN pode ser estruturada em uma topologia estrela ou em variações através de conexões ponto-a-ponto como redes árvore, *mesh* ou anel (figura 3.2). Cada uma das topologias exige a presença de ao menos um dispositivo de função completa.

Na rede ponto-a-ponto um coordenador faz uma conexão direta com um RFD ou outro FFD dependendo da complexidade da conexão. Na rede estrela a presença de um coordenador central é esperada. Esse é conectado com todos os nós restantes da rede.

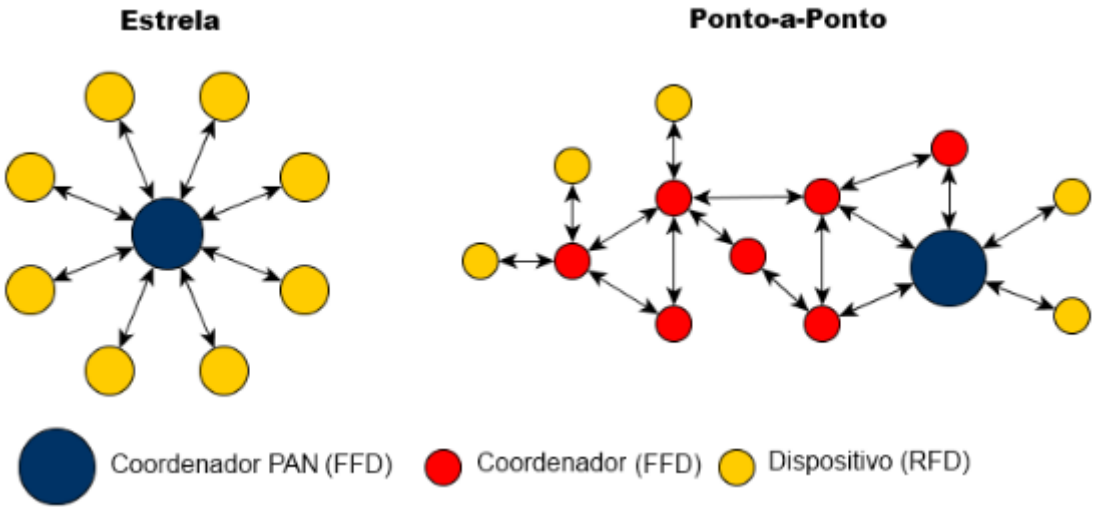

Figura 3.2: Topologias de rede IEEE 802.15.4

3.3.4 ZigBee Alliance

A ZigBee Alliance foi incumbida de especificar as camadas acima das de enlace e física, previamente definidas pela IEEE 802.15.4.

Através das camadas de rede e aplicação o padrão ZigBee insere recursos importantes à redes sem-fio pessoais. Algoritmos de encaminhamento de mensagens fazem que redes mais complexas possam ser montadas com o menor consumo de energia. Alem disso, importantes adições de segurança fazem das redes ZigBee uma importante ferramenta para muitas aplicações críticas.

Aplicado à rede de sensores sem-fio, os padrões definidos pela IEEE 802.15 se tornam muito interessantes, especialmente às especificações criadas pela IEEE 802.15.4 em conjunto com a ZigBee Alliance. Essa tecnologia se encaixa nos conceitos apresentados anteriormente em relação à RSSF.

3.4 MaxStream XBee

Os XBee são módulos receptores e emissores de rádiofrequência microcontrolados que implementam em parte o padrão ZigBee. Desenvolvidos pela empresa *MaxStream*, estão divididos em dois grupos de grupos de produtos: os XBee OEM S1 que cobrem parcialmente apenas as definições feitas pela IEEE 802.15.4, e os XBee ZB que implementam também as especificações feitas pela ZigBee Alliance.

Os XBees OEM S1 versão PRO usados nesse trabalho não implementam rede com roteadores e encaminhamento de mensagens, dessa maneira, apenas as topologias estrela e ponto-a-ponto (simplificada) são suportadas. Cada módulo é genérico, ou seja, pode assumir tanto o papel de coordenador como dispositivo final. Além dessa, uma série de configurações podem ser feitas pelos usuários, e para isso, um protocolo interno de comandos é implementado pelo módulo. Possuem um alcance de 100 m para ambientes fechados e até 1,5 Km para ambientes abertos e sem obstáculos.

Os módulos disponibilizam uma série de pinos para uso das mais variadas aplicações. Entradas analógicas podem conter sensores de temperatura por exemplo. Saídas PWM podem acionar motores DC variando velocidades, e entradas e saídas digitais podem ser configuradas como *trigger* para o envio de mensagens a outros módulos.

Há dois modos de comunicação entre módulos XBee. Modo transparente, em que o dispositivo conectado ao módulo XBee não tem o conhecimento do meio de transmissão, se comporta como se um cabo o estivesse conectando ao destino, ou seja, uma conexão ponto-a-ponto simples. Apenas conexões diretas entre dois dispositivos é possível nesse modo. Por outro lado o modo API permite que a aplicação que a use implemente o protocolo especificado pela MaxStream contendo, por exemplo, o endereço de origem e de destino. Dessa forma uma rede ponto-multiponto pode ser montada.

Tais configurações são acessadas e alteradas através do modo de comandos AT. Para acessar o modo a sequência de caracterees "+++"deve ser enviada ao módulo. Posteriormente qualquer comando especificado no manual do fabricante com seus respectivos parâmetros para a configuração do XBee pode ser enviado.

Para trocar o modo de operação de transparente para API, por exemplo, a seguinte sequência é requerida: "ATAP01". Primeiramente o prefixo "AT"identificando o modo comando, em seguida o parâmetro que se deseja alterar e por fim o novo valor que o registrador referente à configuração ira conter. Nesse caso, o registrador de configuração para o modo de operação, representado pelo dois caracterees "AP", é alterado para o valor 1, o qual define o modo API como o novo modo de operação do XBee.

3.4.1 Modo API

Para a montagem de uma rede estrela, o modo API deve ser configurado no nó coordenador da rede ao menos. Dessa forma é possível direcionar pacotes a módulos específicos assim como detectar a origem de cada informação recebida.

O protocolo a ser implementado é descrito na figura 3.3. O pacote é composto por um byte de início, representado pelo caractere ’˜ (0x7E), por dois bytes contendo a informação de tamanho dos dados encapsulados, os dados propriamente ditos, e um byte de *checksum*.

O primeiro byte dos dados contém o identificador do comando API. A tabela 3.2 apresenta alguns dos comandos presentes na versão S1 do módulo XBee Pro.

De modo a permitir que as configurações do módulo possam ser alteradas no modo API, o identificador 0x08 pode ser usado com os mesmos parâmetros do modo transpa-

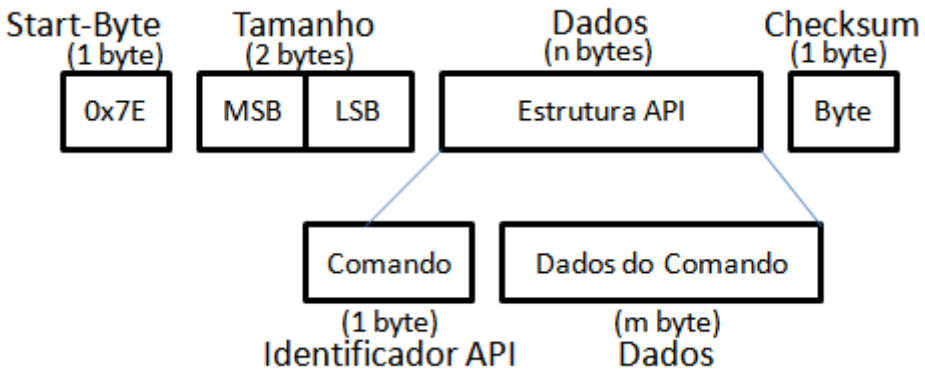

Figura 3.3: Protocolo API

Identificador	Descrição
0x08	Comando AT
0x09	Comando AT (pilha de comandos)
0x01	Transmissão de dados (endereçamento de 16 bits)
0x89	Estado da última transmissão
0x81	Dados recebidos (endereçamento de 16 bits)
0x83	Dados recebidos (Entradas digitais)

Tabela 3.2: Identificadores de comandos API

rente.

Para o envio de dados a outros módulos os identificadores 0x00 ou 0x01 devem ser usados. A figura 3.4 demonstra a sequência de bytes especificada.

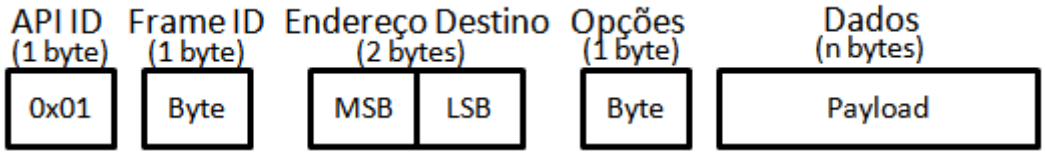

Figura 3.4: Pacote de envio de dados pelo modo API e endereçamento de 16 bits.

A figura 3.5 mostra a sequência de bytes recebida quando dados são transmitidos de outro módulo XBee pelo comando descrito pela figura 3.4.

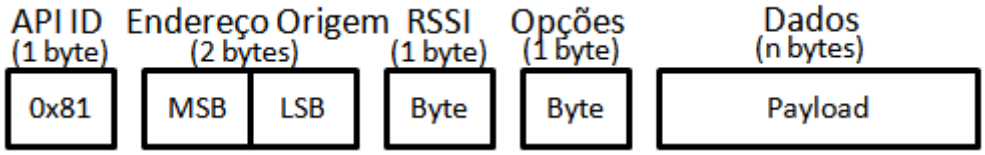

Figura 3.5: Pacote referente ao recebimento de dados no modo API e endereçamento de 16 bits

Ambos os pacotes possuem a limitação de 100 bytes para o tamanho do dado encapsulado.

A estrutura dos dados recebida a partir eventos gerados a partir de mudanças nos estados de discretos (bordas de subida ou descida) em outro módulo XBee é semelhante. A diferença está nos dados, que nesse caso são apenas dois bytes. O primeiro byte representando uma máscara para os discretos considerados válidos para o evento, onde cada bit representa uma entrada digital específica no hardware do módulo, e o segundo que contem o dado propriamente dito, ou seja, um bit representando o estado de cada discreto.

4 VEÍCULOS AÉREOS NÃO TRIPULADOS

4.1 Introdução

Veículos aéreos não tripulados são aeronaves providas de sistemas de controle avançados com a finalidade de realizar vôos sem a presença de um piloto. Pode ser controlado remotamente ou até realizar missões de forma autônoma.

Devido ao avanço de tecnologias relacionadas, e por consequência a diminuição dos custos antes proibitivos dos equipamentos necessários, os VANTs começam a estar presentes em pesquisas científicas e nas mais variadas aplicações.

As primeiras aparições dos VANTs ocorreram na década de cinqüenta com o norte-americano Ryan Firebee, controlado remotamente para fins de reconhecimento de territórios. Porém sua grande inspiração foram as bombas voadoras alemãs tipo V-1, usadas no fim da segunda guerra mundial.

Atualmente os investimentos feitos pelas forças armadas de todo o mundo para o desenvolvimento de VANTs estão cada vez maiores, se tornando um negócio multibilionário. Os Estados Unidos, com o início do combate contra o terrorismo no oriente médio em 2001, duplicou o seu orçamento para pesquisa em VANTs. Segundo dados do próprio departamento de defesa norte-americano, no ano de 2009 cerca de 3,2 bilhões de dólares foram destinados para essa área. O Global Hawk é um exemplo bem sucedido de um veículo aéreo não tripulado. A aeronave, que voa a grandes altitudes, foi amplamente usado em missões de reconhecimento do exército norte-americano no Afeganistão.

Figura 4.1: VANT Hermes450 utilizado pelas forças armadas do Brasil

O Brasil possui um grande interesse dessas aeronaves principalmente com objetivo de monitoras as suas fronteiras, e assim combater o tráfego ilegal de mercadorias e drogas. O Hermes450 (figura 4.1), das empresas AEL Sistemas e ELBIT Systems, está em fase

de avaliação na base aérea de Santa Maria no Rio Grande do Sul, e pode ser usado no futuro pelas forças armadas brasileiras para o patrulhamento das fronteiras do país.

O desenvolvimento dessas aeronaves pode sofrer grandes variações em relação às convencionais. Todos os dispositivos embarcados de interação homem máquina (IHM) e equipamentos que visam o conforto do piloto não são necessários. Com isso, o custo e o peso do projeto são reduzidos de forma substancial. Além disso, os limites do corpo humano que restringiam a desempenho de aviões podem ser ignorados nesse caso. O tempo de vôo antes limitado pela autonomia da aeronave e pelos limites físicos do piloto, é aumentado de forma substancial. Missões de patrulhamento que decorrem por mais de 24 horas podem ser realizados sem a necessidade de uma aterrisagem para reabastecimento por parte do VANT, aumentando a eficiência da tarefa realizada.

Além disso, requisitos de potência e tamanho visando a presença de um piloto a bordo também não é mais exigida. Aplicações onde a aeronave deve se passar por despercebida ou onde o espaço para o deslocamento é extremamente limitado podem se fazer do uso de mini VANTs, que além do tamanho reduzido são muito mais silenciosos. Podem exercer papel crucial também em rede de sensores sem fio como nós móveis, tornando a missão, cuja a rede está inserida, muito mais flexivel.

4.2 Plataforma Skydrones

Figura 4.2: Quadricóptero da empresa *Skydrones*

A Skydrones é uma empresa brasileira situada em Porto Alegre, no Rio Grande do Sul, com foco no desenvolvimento de micro-VANTS. Seus produtos tem sido usados principalmente na área de segurança, e através de câmeras instaladas nas aeronaves, monitorar regiões de interessa de forma a prestar suporte a profissionais em solo.

São helicópteros não tripulados de 4, 6 ou 8 hélices. Possuem uma alta capacidade de estabilização em vôo, e apesar de serem controlados a partir de controles remotos de aeromodelismo, podem ser programados a seguir rotas predefinidas de forma autônoma. Com peso entre 1 Kg e 4 Kg tem autonomia de vôo de até 40 minutos dependendo da bateria empregada e da carga que estão carregando.

Seus VANTs já foram postos em prova em situações reais de segurança. Foi empregado em alguns eventos críticos como jogos de futebol, monitorando possíveis pontos de conflitos entre torcidas rivais. A final da Copa Santander Libertadores de 2010 e um jogo entre as equipes do Sport Club Internacional e do Grêmio Foot-ball Porto Alegrense, que

tradicionalmente apresenta confronto de torcida pela rivalidade entre os dois clubes, são alguns exemplos práticos de uso em conjunto com a Polícia Militar.

Um de seus produtos tem como projeto originário da Mikrokopter, uma companhia alemã que também desenvolve veículos aéreos não-tripulados rádio-controlados. A *Mikrokopter* disponibiliza em seu site boa parte de seus projetos, desde a mecânica como o software, armazenado em repositórios com permissão de leitura ao público. Dessa forma uma grande comunidade de hobbistas foi criada em torno dessa plataforma, auxiliando no seu constante desenvolvimento e aprimoração.

Um quadricóptero, helicóptero de quatro hélices, da Skydrone com projeto da *Mikrokopter* [16] foi usado nesse trabalho.

Apesar de possibilitar o uso de um controle remoto assim como qualquer equipamente de aeromodelismo, esse VANT tem como princpial característica o seguimento autônomo de *waypoints*. Seu software está em constante desenvolvimento, e novos recursos são liberados a cada nova versão gerada pelos desenvolvedores. O *Failsafe*, por exemplo, faz com que o helicóptero volte automaticamente a sua posição inicial, sem a intervenção do operador. É usado em caso de perda de sinal de rádio do controle, e faz com que a aeronave volte a pouse em sua posição de origem.

A partir de um software desenvolvido pela própria *Mikrokopter*, o *Kopter Tool*, é possível a edição e o envio de *waypoints* ao quadricóptero. Para cada ponto, é definido alguns parâmetros além da coordenada geográfica, como, por exemplo, a velocidade de subida (ou descida) e o tempo de espera em cada ponto. Para iniciar o funcionamento do seguimento autônomo na atual versão do software embarcado, o operador deve decolar o helicóptero e posicioná-lo a uma altitude razoável para o seu propósito. Em seguida, através de determinados botões no controle, aciona o modo autônomo, dispensando a partir desse momento o uso do controle remoto. O quadricóptero percorrerá todos o pontos definidos e ficará aguardando por comandos no último *waypoint*.

4.2.1 Hardware

O quadricóptero tem as suas hélices impulsionadas por motores *brushless* que tem como característica a ausência de escovas, o que os diferencia dos motores de corrente continua comuns. Possuem como principial vantagem uma maior eficiência, entregando mais torque por watt que motores elétricos convencionais. Alem disso diminuem consideravelmente a interferência eletromagnética. São largamente usados por modelistas.

O seu projeto não contém nenhum tipo de servo-motor para o controle de sua estabilidade. Esse controle é realizado apenas com a variação da potência destinada a cada motor. O módulo de controle centraliza as informações geradas pelos sensores de giro, de aceleração e de pressão atmosférica para poder ajustar as velocidades de cada motor. A Figura 4.3 mostra uma interface intermediária entre os atuadores e o módulo de controle. Essa interface além de implementar um circuito conversor de potência para o controle dos motores *brushless* também implementa um sistema de detecção *back-EMF*, destinado ao cálculo de posicionamento do eixo do motor, enviando essa informação de volta a controle de estabilidade.

A arquitetura é constituída por quatro camadas como mostrada na Figura 4.4. Os atuadores que fazem o veículo decolar e pairar no ar. Uma interface de potência para o controle dos motores com detecção de posição angular. O controle de estabilidade usando sensores apropriados juntamente com as informações geradas pela interface de potência para ajustar de forma precisa a velocidade de cada motor. E por fim, o sistema de navegação, que, através de sensores de georeferenciamento, controla ângulos de ataque do

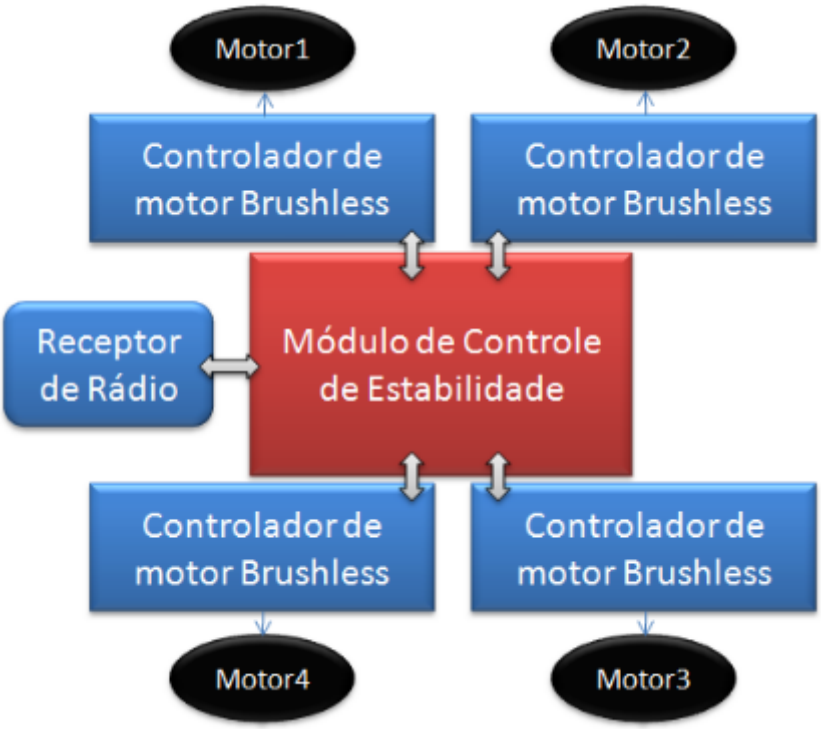

Figura 4.3: Estrutura de hardware para o controle de estabilidade do quadricóptero

VANT para poder realizar o seguimento autônomo de *waypoints* em uma determinada região. Um receptor de rádio de aeromodelismo deve ser conectado ao controle de estabilidade para que movimentos não realizados de forma autônoma, como a decolagem, possam ser feita pelo usuário a partir de um controle remoto.

O módulo de controle de estabilidade possui um microcontrolador 8 bits ATMEGA644 da ATMEL. Acelerômetros de três eixos, x,y e z, giroscópio com 3 graus de liberdade assim como o sensor de pressão atmosférica são acoplados à placa de controle. O condicionamento adequado de sinal para cada tipo de sensor também é implementado no circuito do módulo para que o sinal na entrada do conversor A/D do microcontrolador tenha sensibilidade e resolução adequada para aplicação. A comunicação com a interface de potência é feita através do protocolo I2C, transmitindo a informação de velocidade angular necessária para cada hélice. O VANT pode ser controlado diretamente por um controle remoto comum usado para aeromodelismo através do protocolo PPM. Os parâmetros importantes para a estabilidade da aeronave assim como dados relevantes ao algoritmo de navegação são transmitido via protocolo SPI entre o módulo de controle e o de navegação. Informações usadas para depuração e telemetria no computador são transmitidas via protocolo RS-232.

Devido o alto grau de processamento exigido pelas tarefas destinadas à placa de navegação, um microcontrolador STR91 de 32 bits baseado na arquitetura ARM9E e fabricado pela STMicroeletronics é usado. Possui controladores internos de rede 802.3, USB e protocolo CAN. Possui uma interface com cartões SD usando o protocolo SPI para que dados de vôo sejam gravados ou rotas predefinidas sejam lidas e processadas. Para auxiliar na navegação do quadricóptero dois módulos adicionais foram desenvolvidos pela *Mikrokopter*. Um para o georeferenciamento da aeronave contendo o GPS LEA-6S da *Ublox*, e outro contendo um magnetômetro para o auxílio do correto direcionamento do helicóptero. A interface de comunicação externa com o microntrolador é o padrão RS-232.

Figura 4.4: Estrutura de módulos de hardware completa do quadricóptero

4.2.2 Interface de Comunicação

Um protocolo próprio para acesso de dados do helicóptero assim como configuração do mesmo foi desenvolvida pela *Mikrokopter*. A partir dele é possível requisitar informações como a posição atual do VANT,caso o sinal GPS esteja disponível, ou também o ângulo de inclinação atual dos três eixos de liberdade do helicóptero. O envio de *waypoint* para o seguimento de rotas pré-determinadas também é suportado.

O sistema funciona como uma relação cliente-servidor, em que o helicóptero se comporta como servidor de informações relativas ao seu estado, respondendo requisições do dispositivo conectado à sua porta serial.

Cada módulo de hardware desenvolvido pela *Mikrokopter* possui um endereço lógico e implementa a interface de comunicação. Dessa forma, é possível realizar a comunicação direta com as placas de potência, por exemplo, com objetivo de ler e ajustar determinadas configurações do hardware. Apesar do protocolo ser gerido independentemente, a placa de navegação tem o papel de centralizar os pacotes e enviar aos seus destinos corretos. Para isso, um comando comum para todos os módulos é definido para redirecionar a comunicação do helicóptero, informando ao módulo central o novo destino dos dados recebidos pelo quadricóptero. Para acessar informações relativas aos sensores de estabilidade, por exemplo, um comando deve ser enviado inicialmente requisitando o redirecionando da comunicação para a placa de estabilidade, para que os comandos específicos de requisição de dados de sensores possam ser interpretados corretamente. A partir desse momento, todos os dados recebidos pela placa de navegação são redirecionados automaticamente à placa de estabilidade, sem a verificação de seus dados, tornando necessário outro envio do comando de redirecionamento para os pacotes voltem a ser decodificados no módulo central.

Cada pacote do protocolo é constituído como descrito na Tabela 4.1. Cada módulo possui serviços identificados pelo ID no pacote, o identificador de requisição. De acordo com o ID, o *payload* esperado varia, tanto para a requisição do cliente como para a resposta do servidor. Dessa forma, o tamanho do pacote não está inserido no cabeçalho da mensagem, visto que o tamanho é fixo para cada ID. O comando de requisição cujo iden-

tificador é representado pelo caractere ASCII ’p’ por exemplo, não contém nenhum dado na área de *Payload*. Em compensação, a resposta representada pelo identificador ’P’ contém 22 bytes de dados encapsulados contendo os 11 canais PPM recebidos pelo receptor de rádio de aeromodelismo.

Start-Byte	**Endereço**	**ID**	**Payload**	**Checksum**	**Stop-Byte**
’#’	’a’+ Endereço do módulo	Identificador de requisição	Estrutura de dados em Base64	Checksum incluindo Start-Byte	’\r’

Tabela 4.1: Pacote de dados do protocolo de comunicação criado pela *Mikrokopter*

O *payload* é enviado em Base64, uma codificação que representa um conjunto de dados como caracteres ASCII. Para cada 6 bits de dados um caractere ASCII de 8bits é codificado, numa representação que suporta 64 símbolos distintos. Dessa forma os pacotes enviados são facilmente identificados em um terminal, apresentando todos os bytes da mensagem.

5 IMPLEMENTACÃO

5.1 Ferramentas Utilizadas

Devido aos seus inúmeros pontos positivos e especificações voltadas ao baixo consumo de energia e baixo custo mencionadas no capítulo 3, as rede pessoais locais sem fio (WPANs) implementadas pelos módulos de rádiofrequência da *Maxstream* se mostraram facilmente aplicáveis aos propósitos desse trabalho. Para a montagem da rede sem-fio então, foram usados os módulos XBees OEM Pro S1 da *MaxStream* (figura 5.1). As próprias placas de desenvolvimento dos módulos, com seus recursos, foram empregados como sensores estáticos da rede.

Um quadricóptero da *SkyDrone* (figura 5.2) foi utilizado como nó móvel da rede de sensores. Todos os equipamentos foram cedidos pelo Laboratório de Sistemas de Controle, Automação e Robótica (LASCAR) da UFRGS.

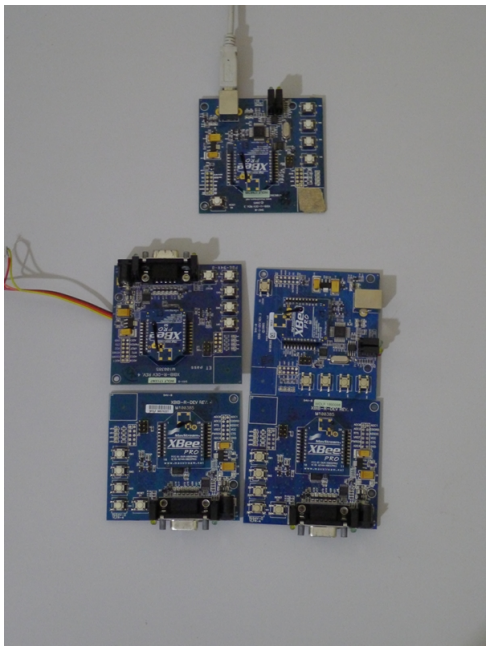

Figura 5.1: XBees e placas de desenvolvimento

Para o desenvolvimento do software, o *framework Qt* da *Nokia* foi usado, utilizando a linguagem de orientação a objetos C++. Além de ser uma ferramenta já usado em outros trabalhos, o *framework* possui a vantagem de ser multi-plataforma, ou seja, facilmente adaptável para outros sistemas operacionais como linux por exemplo. Apesar de não

Figura 5.2: Quadricóptero usado

compartilharem todas as mesmas bibliotecas, é possível também a adaptação do software desenvolvido em Qt Windows para sua versão móvel voltada para celulares.

O Qt facilita também a implementação do padrão *Observer* de orientação a objetos. O padrão *Observer* cria uma relação de dependência entre um método com diversos outros objetos não necessariamente filhos da sua classe. Dessa forma, quando um estado do objeto for modificado por exemplo, todos os seus dependentes serão notificados. Alem disso, mantém uma alta coesão das classes, forçando a definição de uma interface , sem que as mesmas conheçam os métodos e atributos das outras.

As missões da rede são definidas através de uma linguagem de descrição de alto nível estruturada em XML. A MDL (Mission Description Language) [9] define requisitos e restrições que caracterizam o comportamento e a forma de controle de uma rede de sensores sem fio. Além disso, descreve o ambiente presente na rede, especificando todos o nodos estáticos e os pontos de interesse.

A missão descrita é decodificada de forma a gerar parâmetros de configuração do sistema com o objetivo de resgatar informações do ambiente desejadas pelo usuário. A tradução é feita a partir de um analisador sintático, identificando cada elemento da linguagem.

Existem outras formas de configurar missões em RSSFs, como abordagens baseadas em estruturas com banco de dados [15] ou com agentes de software [10]. Um problema comum destas e outras soluções existentes se refere a questão da especificação das missões, uma vez que de forma geral, sao especificadas utilizando linguagens pouco amigáveis aos usuários finais [9]. A MDL, por outro lado, propõe uma maneira para contornar esse problema, fornecendo uma forma mais simples e amigável de descrição, porém mantendo os potenciais de especificação da missão.

A linguagem de descrição contém as informações básicas da missão como nome, descrição e as suas tarefas globais. Pode ser definida uma área de cobertura da missão, descrito por um conjunto de coordenadas geográficas, na qual as ações correspondentes serão tomadas. Especifica também o tempo em que a missão será executada, delimitando

o tempo de início e tempo de término, ou definindo eventos que possam iniciar determinados comandos.

Além de todas as informações básicas relacionadas como as coordenadas de sua localização, em cada nodo ou ponto de interesse, comandos podem ser definidos. Um exemplo de comando a um nodo estático seria, por exemplo, "Se detectado movimento emite sinal". Dessa forma eventos são gerados em cada nó, ocasionando reações por parte da própria rede através de seus nós móveis.

A linguagem *Javascript* foi usada, em conjunto com uma API disponibilizada pela *Google*, para demonstrar pontos de interesse da missão em um mapa através de coordenadas enviadas pelos nós móveis ou definidas na MDL.

5.2 Estrutura do Software

O software é divido em três blocos. Interface com o usuário, onde cada informação é formatada e inserida em uma interface gráfica compatível às necessidade. Interface com a rede de sensores sem-fio, onde os protocolos específicos de cada nodo da rede são tratados, e o gerenciamento de missão que, além de realizar a integração entre os outros dois blocos, comanda e monitora a rede de sensores sem-fio a partir da descrição de missão carregada.

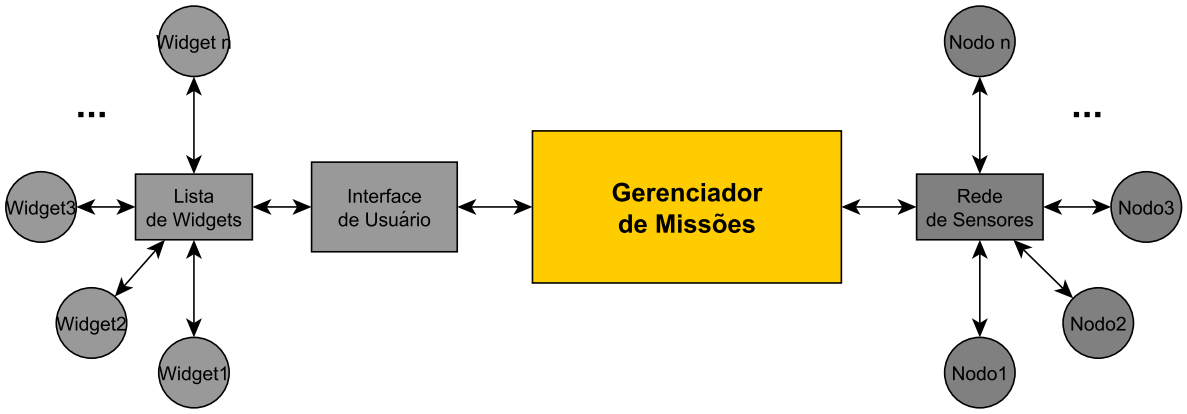

Figura 5.3: Estrutura do Software

A figura 5.3 mostra a composição dos blocos. É possível analisar também, a relação entre os nós da rede e os chamados *widgets* da aplicação. *Widget* é uma abstração para um conjunto de elementos gráficos contendo informações para a interface com o usuário. Tem por finalidade além do reuso de código, a padronização da forma como os dados sao manipulados. Dessa maneira, cada nó da rede de sensores está representado por um *widget* para a visualização do operador.

O gerente da missão contém todas as informações referentes à missão, como a sua descrição e suas fronteiras geográficas por exemplo. Possui o papel também de roteador de mensagens da rede, repassando cada dado ao seu destino correspondente na estrutura. Os dados enviados pelo Nodo *n* devem ser corretamente traduzidos e, se constatada a validade do pacote, o gerenciador deve encaminhar a informação ao *Widget n*, via interface de usuário. O fluxo contrário também deve ocorrer, cada nó da rede de sensores deve receber a sua requisição correspondente.

A classe *NetworkMission* implementa o gerente de missão como mostrado no diagrama de classes da figura 5.4. Herda os recursos da classe *QObject* do próprio Qt, para que possa implementar o padrão *Observer*. Dessa forma permite a criação de uma interface em que alguns de seus métodos podem ser chamados pelas suas classes filhas. Essa característica faz com que o uso e a criação de eventos dentro da própria estrutura do software seja possível, e consequentemente, a implementação de protocolos assíncronos pode ser realizada.

Para a informação de posicionamento geográfico de cada nodo, são herdados os recursos da classe *GPSPosition*, que possui os atributos usados para a geolocalização: latitude, longitude e altitude. Define métodos de tradução de coordenadas passadas por diversos formatos, como strings contendo a representação de graus, minutos e segundos por exemplo.

Para criar a rede de sensores, a *NetworkMission* possui como atributo a *missionNodesList*. É uma lista que contém todos os nós adicionados à rede, tanto móveis como estáticos. Para isso, é usado o conceito de *polimorfismo* através da classe abstrata *MissionNode*. Dessa maneira, a forma com que cada nó trata seus dados faz com que a implementação da classe de gerência da missão independa das interfaces utilizadas por cada dispositivo anexado ao sistema. Porém, para realizar a distribuição correta de pacotes de dados, uma tabela de tradução *translateCommunicationType* faz-se necessária.

A classe *MissionNode* contém todas os atributos e métodos necessários que todos os nós devem herdar para que possam ser monitorados e comandados pelo gerente de missões. Possui um método virtual chamado *dataHandler*, porém, apenas o seu protótipo está declarado. Por ter a característica de ser virtual, o compilador obriga que todas as instâncias que herdam a classe tenham localmente a sua implementação, de acordo com as suas variações de atributos e protocolos. Para a implementação de eventos e interfaceamento com o gerenciador de missões, herda também a classe *QObject*.

A classe *ZigBeeTransparentStaticNode*, que herda a classe *MissionStaticNode*, que por sua vez herda os recursos de *MissionNode*, implementa um nó sensor estático de um discreto, ou seja, uma entrada digital presente em um módulo XBee. Pode abstrair, por exemplo, o comportamento de um sensor de presença ou de movimento. O atributo *discrete* armazena o estado atual do fenômeno físico que o sensor está capturando e seu valor é alterado através do método *dataHandler*, cujo parâmetro de entrada é o conjunto de dados enviados pelo nó real da rede. Nessa implementação, a estrutura enviada contendo a informação do discreto é decodificada pelo próprio *dataHandler*.

HelicopterHandler abstrai um no móvel da rede através da herança do *MissionNode*, e em conjunto com a classe *MKProtocol*, implementa a interface com o quadricóptero *Mikrokopter* da *Skydrones*. Internamente, a classe *HelicopterHandler* possui todos os atributos relacionados ao helicóptero, assim como uma máquina de estados para o controle das requisições de dados à aeronave. Seu funcionamento será detalhado na seção 5.2.2.

O gerenciador de missões possui também uma lista de pontos de interesse (*missionWaypointsList*) abstraídos pela classe *MissionWaypoints*. Por possuir atributos semelhantes aos dos nodos da rede como, por exemplo, nome, descrição e posição geográfica, herda também os recursos da classe *MissionNode*.

A interface com a porta de comunicação externa onde os nodos reais da rede são conectados, é implementado pelas classes *SerialHandler* e *ZigBeeProtocol* usando o padrão RS232 e o protocolo API dos XBee respectivamente.

A classe *MainWindow* implementa as interfaces com o usuário, enquanto que a *Mis-*

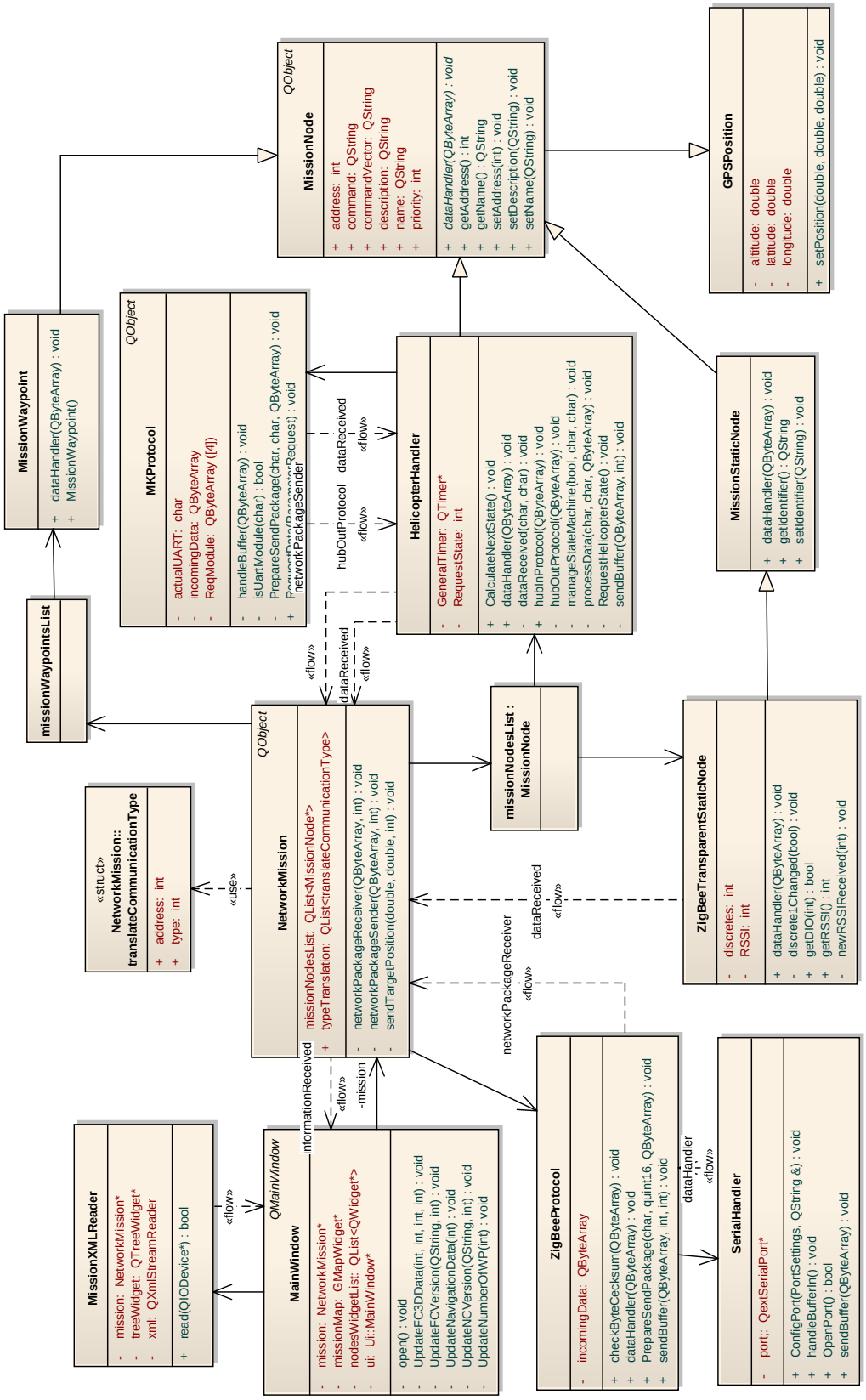

Figura 5.4: Diagrama de classes

sionXMLReader abre, decodifica e prepara o ambiente de missão a partir de um arquivo MDL.

5.2.1 Interface com a rede de sensores

A figura 5.5 mostra as camadas utilizadas pelo software com objetivo de implementar os protocolos de comunicação com os nós da rede.

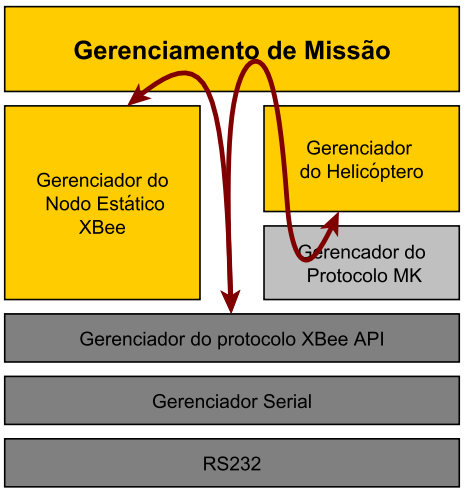

Figura 5.5: Camadas de abstração de comunicação do Software

Na base da pilha de protocolos está a camada do padrão RS232 visto que a conexão entre o computador e o módulo XBee, que desempenha o papel de coordenador na rede, é feita através de uma porta serial comum do sistema operacional.

Serial Handler provê, em conjunto com uma biblioteca amplamente usada pela comunidade Qt chamada *QExtSerial*, a interface de mais baixo nível com a porta de comunicação serial. Através dela, todos os dados endereçados a essa porta são enviados ao software para decodificação, assim como os pacotes montados destinados ao meio físico. Encapsula o protocolo API dos módulos XBee para a realizar a comunicação com os todos os nós da rede que usam também módulos Xbee para envio de dados.

Os módulos que representam os nodos estáticos e dinâmicos utilizam o protocolo API, porém implementam as suas próprias estruturas de comunicação. Para o gerenciamento dos pacotes recebidos ou enviados pelo XBee coordenador, o gerente de missão realiza uma verificação de endereços de origem, no caso do recebimento de pacotes, e de destino dos dados, no caso do envio de pacotes. Através de uma tabela de registros, implementada pela estrutura *translateCommunicationType*, verifica qual protocolo está relacionado ao endereço, e dessa forma, direciona o pacote para o tratamento correto.

O trecho de código da listagem 5.1, mostra a forma com que o redirecionamento dos pacotes de dados é feita. A lista de nodos é percorrida até que o nodo com o mesmo endereço do pacote recebido é encontrado. O método de tratamento dos novos dados de entrada é então chamado.

```
1 void NetworkMission::networkPackageReceiver(QByteArray data, int address)
2 {
3     for(int i = 0 ; i < missionNodesList.length(); i++)
4     {
```

```
5           if (missionNodesList[i]->getAddress() == address)
6           {
7               missionNodesList[i]->dataHandler(data);
8               break;
9           }
10      }
11  }
```

Listing 5.1: Redirecionamento dos pacotes a cada nodo

O diagrama de sequência representado pela figura 5.6 traz o fluxo de dados entre os objetos, mostrando os passos da decodificação de um pacote proveniente de um nodo estático XBee da rede de sensores.

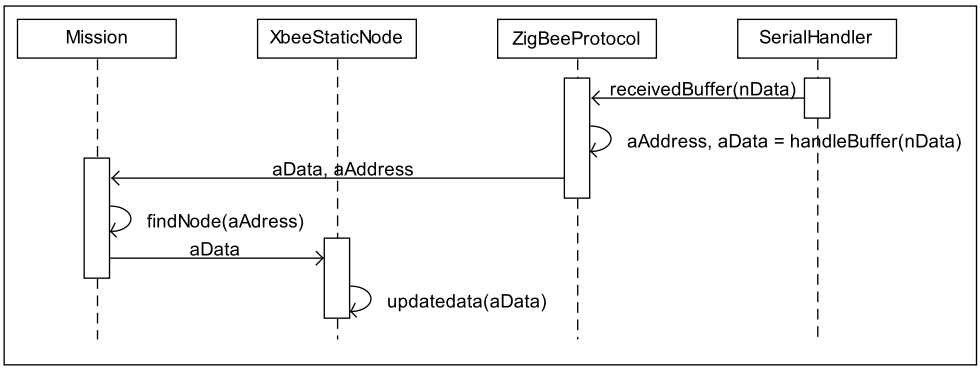

Figura 5.6: Diagrama de sequência para um dados proveniente de um sensor estático

Ao receber um dado pela porta serial do sistema operacional, *SerialHandler* dispara um evento ao objeto de análise do protocolo API. Esse, por sua vez, decodifica o pacote e calcula o *checksum* com o objetivo de validar os dados recebidos. Gera, a partir disso, outro evento, dessa vez ao gerente de missões. Com o endereço recebido, o gerente identifica o protocolo e redireciona ao destino correto. O objeto abstrato nodo estático recebe os dados, os decodifica e atualiza seu atributos internos.

5.2.2 Interface com quadricóptero

Devido à complexidade da interface de comunicação com o quadricóptero, uma classe auxiliar destinada apenas a lidar com o protocolo da *mikrokopter* foi utilizado. Dessa forma, o objeto que abstrai o helicóptero se torna mais coeso e torna transparente a forma em que a troca de dados entre a aeronave e o software é feita.

A classe *MKProtocol* implementa a decodificação, montagem e codificação dos pacote do protocolo. Identificando os campos da mensagem definidos na tabela 4.1.

Como visto na seção 3.4.1, o protocolo API possui a limitação de 100 bytes para o tamanho dos dados encapsulados. Por isso, é inevitável a quebra de determinados pacotes maiores para que tal restrição seja respeitada. A classe *MKProtocol* fica incumbida então, além de codificar e decodificar o *payload* do pacote na codificação Base64, de lidar com parcelas de pacotes e realizando a montagem dos mesmo, de forma a decodificar os dados corretamente.

Além desse serviço, também deixa transparente ao objeto quadricóptero o envio do comando de redirecionamento das requisições a cada módulo da aeronave. Essa tarefa representa o maior *overhead* à comunicação entre o helicóptero e a estação de solo. Enquanto o software do módulo embarcado não efetivar a troca do estado de redirecionamento interno dos pacotes de dados do quadricóptero, qualquer requisição não será propriamente decodificada pelo VANT.

Testes práticos realizados mostraram que um tempo maior que 70ms deve ser respeitado entre o envio do comando de redirecionamento e a nova requisição. Tempos menores que esse fizeram com que pacotes fossem perdidos, já que o helicóptero não possui uma fila de entrada para esse caso, descartando os dados nesse período.

Entretanto problemas enfrentados relacionados à esse comando nao foram solucionados compleamente por essa implementação. O não envio de uma resposta a respeito do seu estado de redirecionamento acarreta na nao sincronia entre os softwares. Esse problema pode acarretar a uma discrepância entre o quadricóptero e a estação de solo. O software pode, por exemplo, iniciar um processo de requisições de dados de *waypoints* à placa de navegação, enquanto que a aeronave pode estar redirecionando todos os pacotes à placa de estabilidade, que não possui a capacidade de decodificar os dados de *waypoints*, acarretando o descarte da mensagem. Esse comportamento levaria a uma falha de comunicação sem a possibilidade de sua recuperação, a menos que ambos os dispositivos, helicóptero e estação de solo, fossem reiniciados.

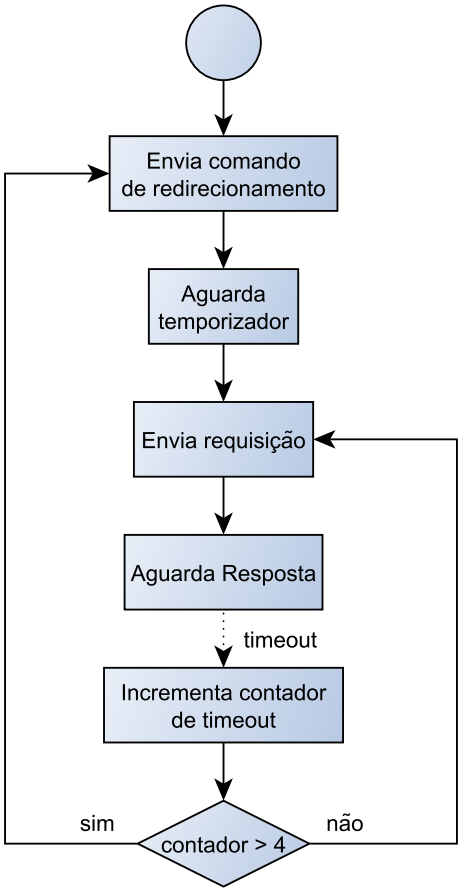

Figura 5.7: Fluxograma da lógica de timeout para o protocolo da *Mikrokopter*

Para diminuir o impacto dessa limitação, uma lógica simples de timeout foi criada. A

figura 5.7 mostra o fluxograma implementado.

Inicialmente identificada a necessidade do redirecionamento da requisição, o comando é enviado ao helicóptero. Um tempo mínimo de aproximadamente 70ms definido por testes é respeitado para que então a requisição propriamente dita seja enviada. Caso a resposta correspondente a requisição não seja recebida em um determinado tempo, um evento de *timeout* é gerado. Um contador é incrementado nesse momento, e enquanto ele for menor que 4, a mesma requisição é reenviada. Caso o mesmo comportamento persista, e o contador passe o valor pré-estabelecido, o processo de redirecionamento de dados é reiniciado, assumindo que o quadricóptero não realizou como o esperado o comando de redirecionamento.

O contador é implementado para separar diferentes diagnósticos de perda de pacotes. Devido a não confiabilidade absoluta do meio físico de comunicação, perdas eventuais de pacotes são esperadas, e são tratados com um *timeout* simples. Entretanto a falta de respostas para seguidas requisições pode se tratar de outro problema, no caso, o não roteamento interno correto dos pacotes recebidos pelo quadricóptero.

O envio do comando de redirecionamento, assim como o temporizador, é implementado pela classe *MKProtocol* e fica transparente à classe *HelicopterHandler*. O método de envio de novas requisições está disposta na listagem 5.2.

```
1  void MKProtocol::RequestData(ParameterRequest Setting)
2  {
3      if(!isUartModule(Setting.getDestDevice()))
4      {
5          if (Setting.getDestDevice() == NC_ADDRESS)
6              sendBuffer(getRequestUartRedirect(Setting.getDestDevice()));
7          else
8              PrepareSendPackage(REDIRECT_UART_HEADER,getActualAddress(),
9                                  getRequestUartRedirect(Setting.getDestDevice()));
10
11         this->setUartModule(Setting.getDestDevice());
12
13         Sleep(WAIT_REDIRECT_TIME);
14     }
15
16     PrepareSendPackage(Setting.getAttributeType(),Setting.getDestDevice(),
17                        Setting.getRequestParameter());
18 }
```

Listing 5.2: Envio de requisições ao quadricóptero

Na linha três é verificada a necessidade do envio do comando de redirecionamento, checando o endereço do módulo de destino do pacote a ser enviado.

O protocolo define dois tipos de pacotes diferentes para realizar a operação. O primeiro para o caso da mudança do módulo de navegação para os demais módulos e o segundo para o caso contrário.

Para o segundo tipo de pacote um conjunto de dados específicos deve ser enviado ao quadricóptero de modo a redirecionar os dados novamente para o módulo de navegação. Como esse pacote deve infringir a forma definida pelo protocolo para o seu cabeçalho, o método tradicional da classe para o envio de dados à aeronave nao pode ser utilizado, e a

chamada direta do método de envio de dados deve ser realizada.

Após esse processo, na linha onze, é atualizado o estado atual de redirecionamento de dados da aeronave, e na sequência o envio propriamente dito da requisição é realizado.

As implementações dos *timeouts* são realizadas pela classe *HelicopterHandler* com o auxílio da classe *QTimer* do próprio framework.

O diagrama de classes da figura 5.8 mostra a estruturação dos objetos com a finalidade de implementar cada tipo de requisição e sua respectiva estrutura de dados.

A classe *HelicopterAttribute* implementa todos os métodos e possui os atributos necessários para a criação e tradução de requisições para o helicóptero. Através da classe *ParameterRequest*, possui os três atributos básicos para o interfaceamento com o quadricóptero. *AttributeType* identificando a requisição que será feita, *DestDevice* identificando o módulo interno da aeronave que os dados devem ser enviados e o vetor de *bytes RequestParameter* que contém os dados propriamente ditos. Através apenas desse conjunto de informações que a *MKProtocol* gera requisições.

Para preencher os dados da requisição, a classe *HelicopterAttribute* é especializada em diversas outras que implementam métodos específicos pra determinadas estruturas de dados.

A classe *VersionInfo* por exemplo, implementa a interface com o serviço de versões de software provido pelo quadricóptero, tanto pela placa de navegação como pela placa de estabilização. A classe é instanciada duas vezes pela *HelicopterHandler*, uma abstração para cada placa. Seu construtor, assim como todas as outras classes que herdam *HelicopterAttribute*, preenche os atributos de acordo com a sua especificação, que no caso é o caractere 'v' para o tipo de requisição e o valor 1 para o endereço da placa de estabilização ou 2 para a de navegação.

Para implementar o protocolo da *Mikrokopter* de forma completa, uma interface é definida entre as classe *HelicopterHandler* e *MKProtocol*. O envio de dados é feito através do método público *RequestNewData*, e o seu parâmetro é dado pelo próprio objeto *ParameterRequest*. O pacote então é propriamente codificado e montado na estrutura definida pelo protocolo. Um evento é gerado, e o pacote repassado de volta ao *HelicopterHandler* que o envia diretamente ao gerente da missão. Dessa forma são criadas classes com alta coesão, permitindo que mudanças no formato das requisições, por exemplo, sejam feitas de maneira independente da implementação do protocolo.

A figura 5.9 contém um diagrama de sequência com as trocas de mensagens entre os objetos do software com objetivo de montar o pacote de requisição para o envio ao quadricóptero.

Um temporizador inicia o processo da criação do protocolo, enviando os atributos da requisição desejada. *MKProtocol* verifica a necessidade do envio do comando de redirecionando e monta a mensagem correspondente. Os dados são enviados ao gerente da missão que determina o tipo do nodo da missão que a mensagem será destinada. De acordo com os parâmetros recebidos, o objeto de tratamento do protocolo API cria o pacote final que será enviado a módulo XBee emissor.

De forma semelhante a de nodos estáticos (figura 5.6), o recebimentos das respostas às requisições é iniciado a partir de um evento gerado pela porta de comunicação serial. A classe *ZigBeeProtocol* identifica o protocolo API, validando o pacote através do *checksum* e de certas consistências exigidas pelas especificações do próprio protocolo.

Através do endereço, o gerenciamento da missão localiza o nodo abstrato correspondente, checa a combinação do tipo de pacote recebido com o tipo do nodo, e o envia ao determinado objeto. Os dados são repassados ao tratamento do objeto que decodifica o

Figura 5.8: Estrutura de atributos para o protocolo da *Mikrokopter*

protocolo do helicóptero, para que os dados extraídos sejam usados para atualizar o estado atual do helicóptero.

Além do *VersionInfo*, outros objetos que implementam atributos do helicóptero foram criados, todos herdando os recursos da classe *HelicopterAttribute*. O tratamento de

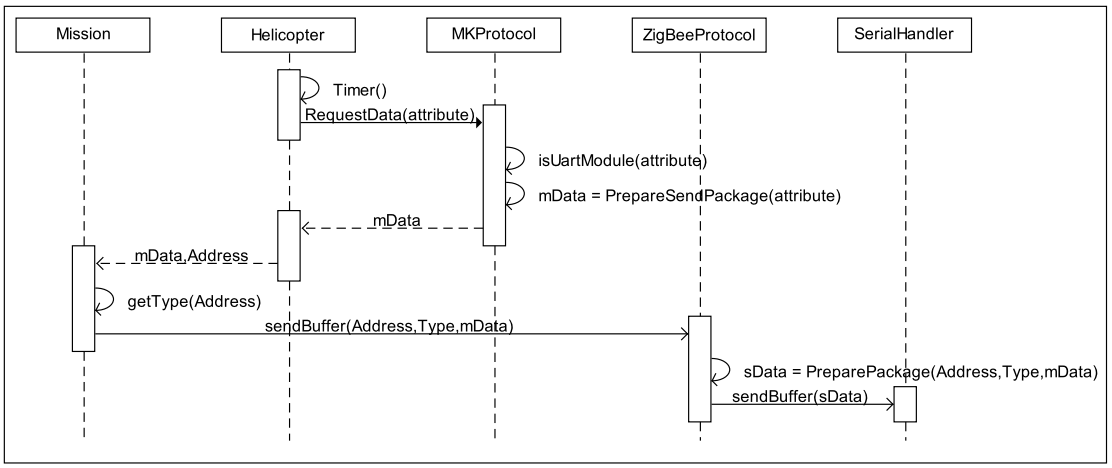

Figura 5.9: Diagrama de sequência para envio de requisições no protocolo da *Mikrokopter*

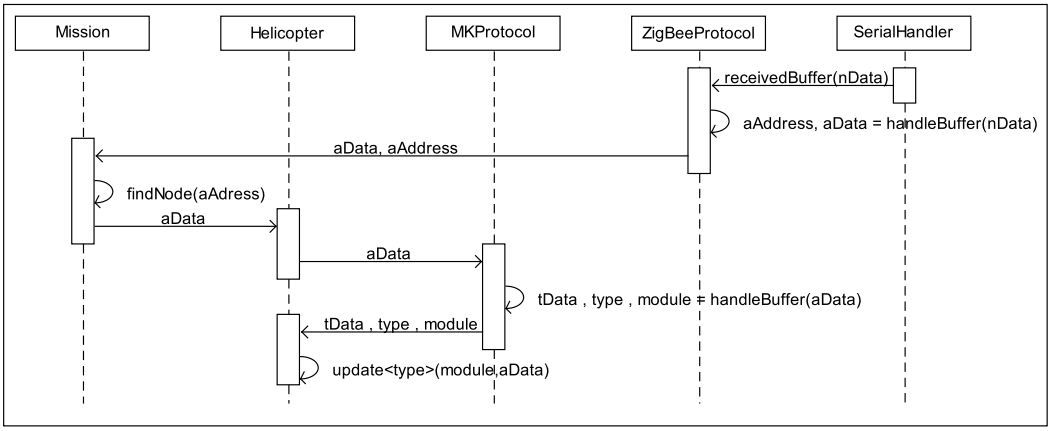

Figura 5.10: Diagrama de sequência para a decodificação de pacotes do protocolo da *Mikrokopter*

pontos de interesse, nos quais o helicóptero deverá percorrer, é feito a partir da classe *WaypointsHandler*. Uma lista de *Waypoints* é mantida sincronizada com o helicóptero, permitindo ao gerente da missão uma agilidade maior, visto que a requisição de determinado *waypoint* interno do helicóptero poderia gerar atrasos nas tomadas de decisão da rede de sensores. Possui métodos de criação de requisições de envio de novo ponto de interesse, limpeza de ambas as listas de *waypoints* (local e embarcada), envio de coordenada destino, além de requisições de *waypoints* já armazenados no quadricóptero.

O atributo *Data3D* adiciona uma característica que o diferencia das demais apresentadas anteriormente, em que, para cada requisição um resposta é esperada. Realiza a requisição dos dados capturados pelos sensores da placa de estabilização. De maneira que são informações extremamente dinâmicas da aeronave, como o ângulo de ataque por exemplo, o helicóptero envia continuamente diversos pacotes durante cinco segundos. A periodicidade do envio é definida pela requisição realizada pelo nó coordenador.

DebugAnalogs também é do tipo periódico, e implementa a requisições de informações como a tensão da bateria presente no helicóptero, a potência em cada motor e a altitude. Com a mesma característica a classe *OSDData* implementa as informações de

coordenada atual da aeronave, número de *waypoints* armazenados, velocidade além de outros atributos.

A requisição implementada pela classe *EngineTest* não espera nenhuma resposta por parte do quadricóptero. Com ela é possível ligar os motores em determinada potência independentemente dos dados lidos dos sensores. É usado para teste dos motores do helicóptero em solo.

A classe *HelicopterHandler* implementa uma máquina de estados para manter na medida do possível todos os seus atributos atualizados em relação ao helicóptero. A figura 5.11 mostra as transições de cada estado.

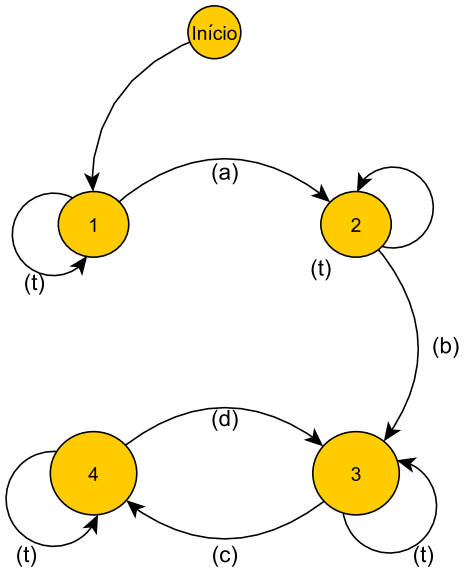

Figura 5.11: Máquina de Estados para Requisição de Informações do Quadricóptero

Nos dois primeiros estados ocorrem as requisições de versões de software embarcado do módulo de navegação e estabilidade respectivamente. Em ambos, a transição é realizada assim que a resposta esperada de cada envio de dados é recebida.

O estado três usa a classe *OSDData* para obter a informação de coordenada atual do VANT e o quarto estado requisita os dados de sensores.

A transição (d) é feita através de um temporizador de dois segundos, ou seja, são decodificados todos os pacotes enviados periodicamente pelo helicóptero durante esse tempo, para que na sequência um nova coordenada seja obtida. Apesar da requisição do terceiro estado compartilhar da mesma característica de respostas periódicas, a sua transição (c) é feita assim que a primeira mensagem e recebida, visto que não há a necessidade de uma alta taxa de atualização da posição atual do helicóptero.

As versões são adquiridas apenas uma vez, já que não há a possibilidade de atualização de firmware em vôo.

Para o envio de *waypoints* ao helicóptero, uma outra máquina de estados foi implementada a parte. Tendo em vista que o número de pontos de interesse de uma missão para atribuição a nós móveis é alto, causando uma considerável sequência de trocas de mensagens entre a estação de solo e o helicóptero, foi necessário que as demais requisições fossem interrompidas durante o processo. Com isso, a cada envio de uma lista de *waypoints*, a máquinda de estados normal do sistema para a captura constante dos dados

da aeronave é preemptada, de modo que todas as mensagens de envio de coordenadas seja feito sem interferências de outras requisições.

Testes com uma implementação concorrente do envio de *waypoints* foram feitos, e mostraram um comportamento indesejado do sistema, causando demasiada lentidão devido às trocas de redirecionamento interno de mensagens.

5.2.3 Interface com usuário

A classe *MainWindow* tem por objetivo fazer a interface entre o usuário e o gerente de missões. Implementa as funcionalidades apresentadas no diagrama de casos de uso apresentado na figura 5.12.

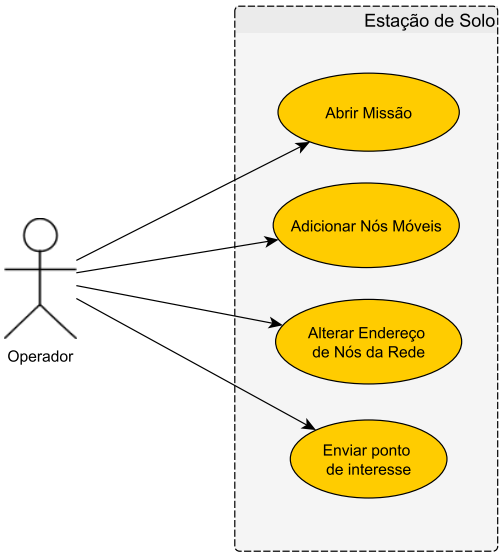

Figura 5.12: Caso de Uso

O operador pode abrir um arquivo de missão MDL, possibilitando à estação de solo configurar e criar o ambiente de controle e monitoramente da rede de sensores sem fio. Adicionar nós móveis, alterar o endereço de nós da rede para que a comunicação seja feita corretamente, e o envio de pontos de interesse.

O diagrama de classes da figura 5.13 mostra a estrutura do software para a interface com o usuário que e composta pelo leitor de MDL, telemetria e controle dos nós da rede e um mapa para o acompahamento da missão.

A classe *GMapWidget* implementa a interação com uma página *web* através do *QWebPage*, uma classe disponibilizada pelo *framework* para inserir na interface uma página da internet.

A página inserida contém um conjunto de funções *javascript* para utilizar a biblioteca do *Google* destinada ao seu software de mapas *Google Maps*. Com isso e possível usar todo o seu extenso banco de dados de mapas com objetivo de mostrar todos os nodos da rede de sensores sem fio na sua real localização.

Uma pequena API *javascript* foi criada para simplificar a implementação da *GMapWidget*. Uma estrutura de nodos semelhante a utilizada pelo gerente da missão foi criada. Um lista genérica de marcadores nós estáticos e móveis foi implementada contendo as informações básicas de cada um: nome, endereço e coordenada. Funções para adicionar tais marcadores ao mapa foram disponibilizadas, de maneira que, através da chamadas dessas

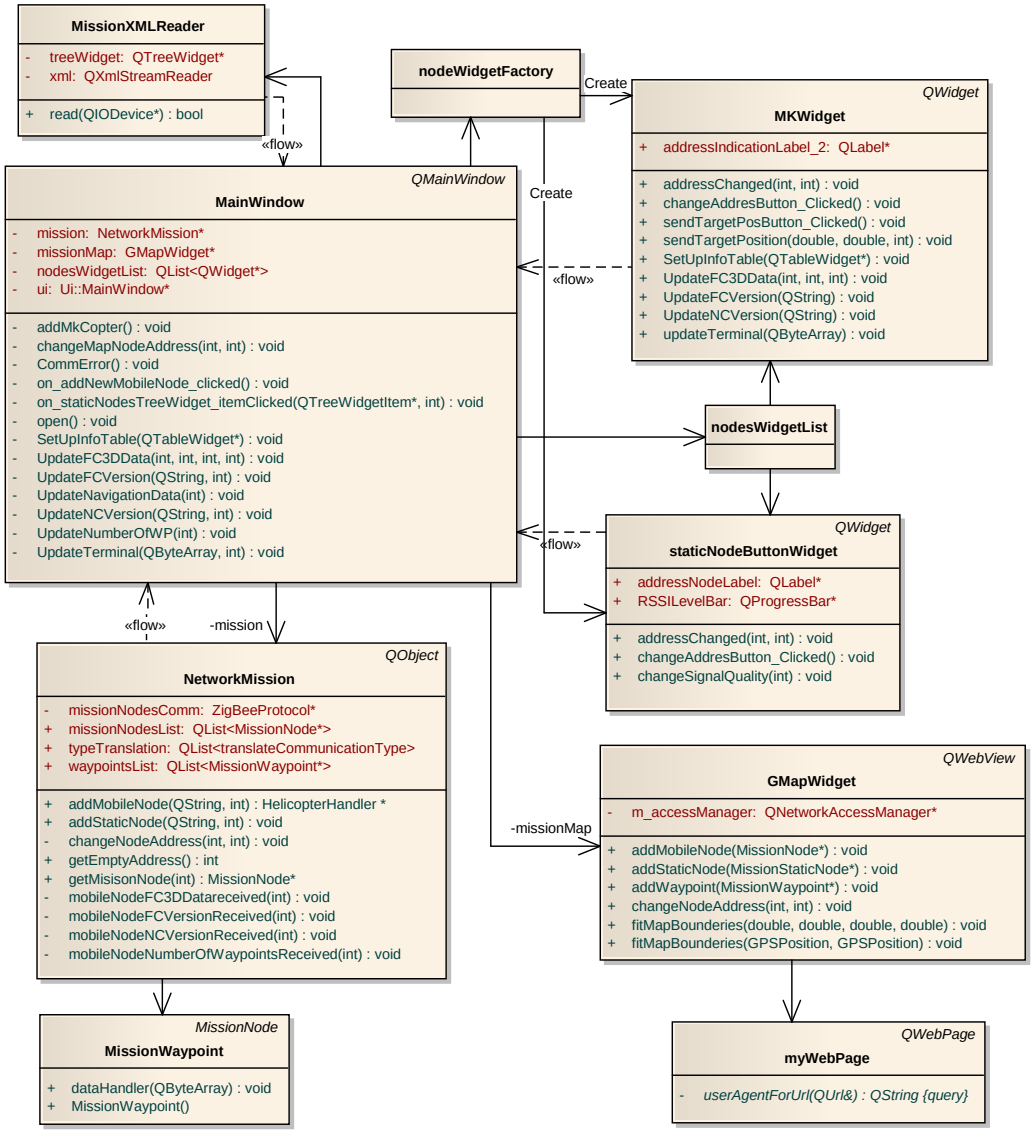

Figura 5.13: Estrutura dos Widgets

funções *javascript* por parte do software desenvolvido em Qt, a disposição do mapa fosse controlada pelo gerente de missão.

A figura 5.14 mostra o exemplo de um mapa mostrando alguns nodos estáticos em verde e pontos de interesse em azul.

Uma lista de *widgets* é mantida pela classe *MainWindow* sincronizada com a lista de *MissionNodes* presente no gerente de missão. Assim como na *NetworkMission*, a lista contém objetos que especializam uma classe comum, criando formas específicas de tratar os dados.

Para esse trabalho foram implementados dois *widgets*: *MKWidget*, contendo a telemetria de um quadricóptero da *Mikrokopter*, e *staticNodeButtonWidget*, com as informações recebidas por sensores estáticos de discretos implementados pelos *XBees*.

MKWidget cria uma interface com três abas. A primeira contendo uma tabela com as versões de software, coordenada atual do helicóptero e a coordenada destino, mostradores de horizonte artificial e direcionamento, e caixas de edição de coordenada destino e

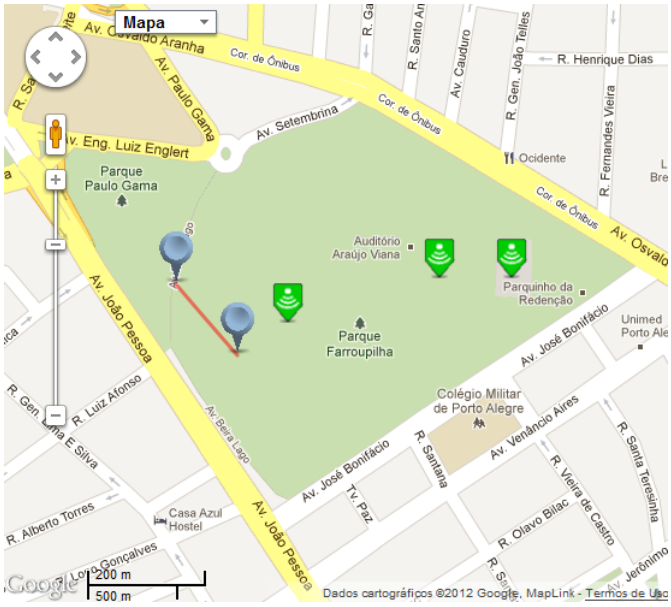

Figura 5.14: Mapa gerado com o *Google Maps*

endereço do nodo móvel. Uma barra indica a atenuação do sinal transmitido. A segunda aba mostra todos os dados recebidos pela estação de solo com o quadricóptero representado pelo *widget* como origem. E a terceira contem um campo para teste dos motores do helicóptero, tendo a possibilidade de acionar cada um individualmente da potência desejada.

A classe *staticNodeButtonWidget*, muito mais simples, cria apenas duas abas. A primeira contendo a caixa de edição do endereço do nodo e segunda com o indicador de atenuação do sinal e um texto indicando se o discreto está em nível lógico alto ou baixo.

As figuras 5.15 e 5.16 mostram a implementação visual de ambos os *widgets*.

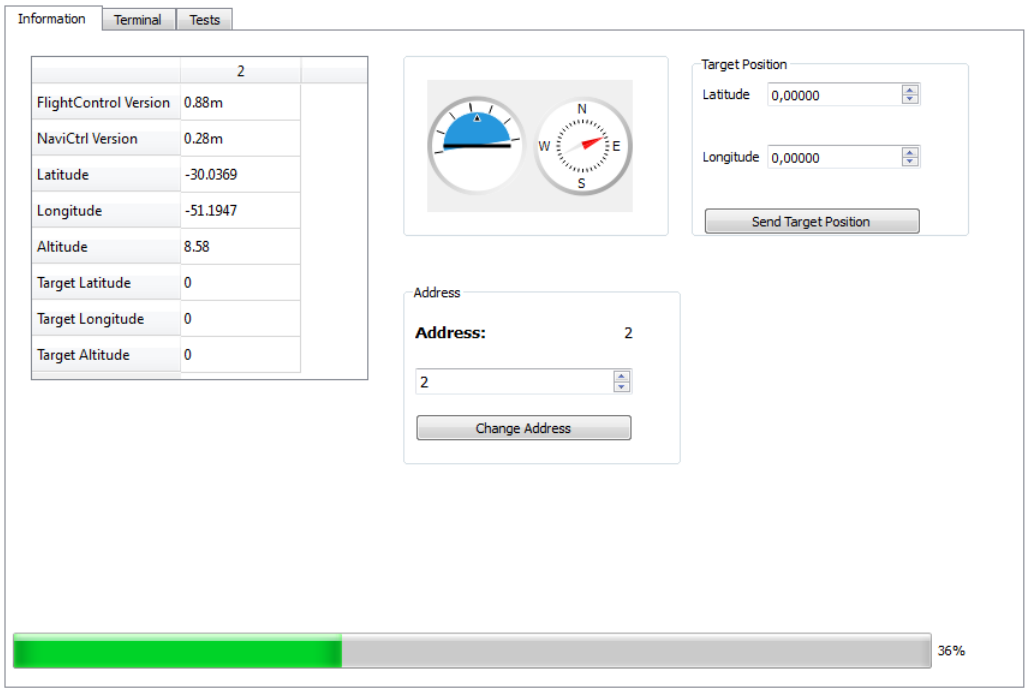

Figura 5.15: Interface criada pela classe *MKWidget*

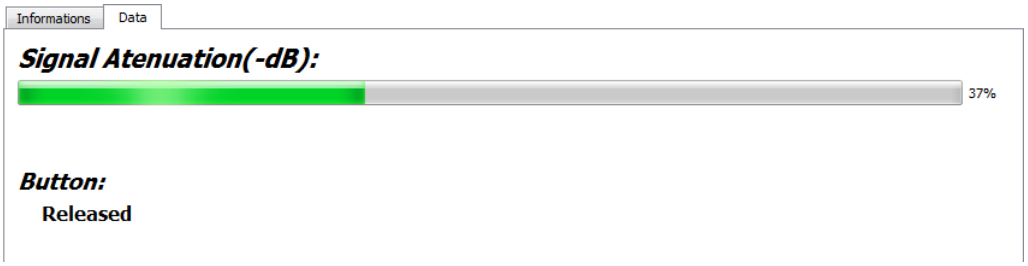

Figura 5.16: Interface criada pela classe *staticNodeButtonWidget*

MissionXMLReader possui os métodos para ler o arquivo MDL e decodificar o seu conteúdo. A partir disso, cria o ambiente para que o gerenciamento da missão seja feito. O fluxograma da figura 5.17 mostra os passos para a decodificação do XML estruturado como uma MDL.

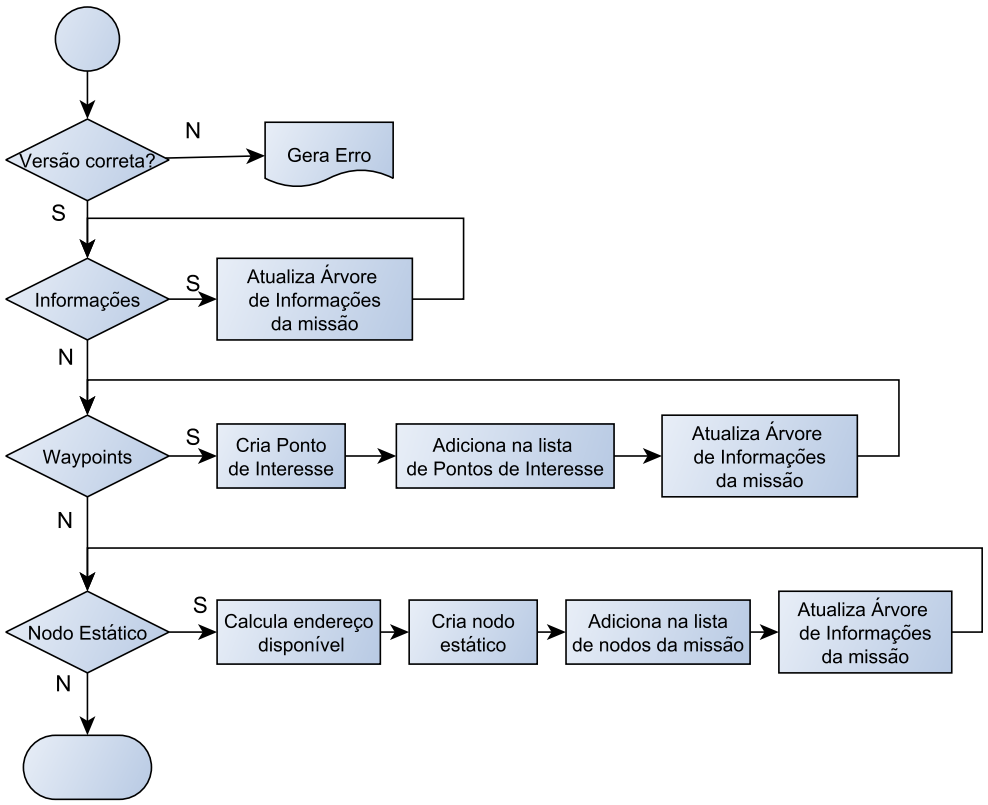

Figura 5.17: Fluxograma para leitura do arquivo de missão

Inicialmente é feita a checagem da versão da MDL definida na primeira *tag* do arquivo XML, dentro do atributo *tool_version*. Garantindo assim a estrutura esperada pelo decodificador. Caso a versão seja incorreta, um erro é gerado para usuário, e o ambiente da missão não é criado.

A classe espera então umas das três *tags* básicas da linguagem de descrição: *information*, *waypoints* e *static_nodes*. *Information* contém todas as informações básicas em relação a missão como a sua descrição, nome e os limites geográficos.

Para cada ponto de interesse presente na missão há uma *tag waypoint*. Contém o nome do *waypoint*, a sua descrição, sua prioridade dentro da missão, a coordenada geográfica, o tempo em que deve ser monitorada. Durante a leitura dos campos, a lista de pontos de

interesse do gerente de missão é preenchida, assim como os respectivos marcadores são inseridos no mapa de visualização geográfica da missão.

Da maneira semelhante, os nodos estáticos são descritos no XML, contendo o seu identificador, sua descrição e coordenada geográfica. Como outros nodos, tanto estáticos e móveis, podem já terem sido inseridos à rede de sensores, é necessária a verificação de endereços disponíveis na lista de nodos. Já que o redirecionamento é baseado basicamente nessa informação, os endereços devem permanecer únicos. Assim como os *waypoints*, são inseridos na lista de nodos do gerente da missão durante a leitura dos valores contidos na MDL, assim como o marcadores no mapa de visualização.

Assim que todo o arquivo de descrição foi processado, os *widgets* são criados de acordo com a estrutura de cada nodo presente na lista do gerente de missões. O padrão *factory* foi empregado para a implementação desse recurso.

O padrão define o uso de "fábricas"de objetos. Sua função é criar*widgets* genéricas, porém especializadas de acordo a estrutura requisitada inicialmente. Dessa forma todos os objetos são "fabricados"da mesma maneira, e são incluídos em uma mesma lista única na classe *MainWindow*, facilitantando o seu gerenciamento.

A integração estre as informações provenientes dos nós da rede recebidas pelo gerenciamento da missão e os *widgets* e seus componentes da interface de usuário é feita pela classe *MainWindow*.

A classe *MainWindow* realiza a integração dos eventos gerados pelo gerente da missão com os blocos de visualização de dados representados pelos *widgets*.

5.3 Demonstração

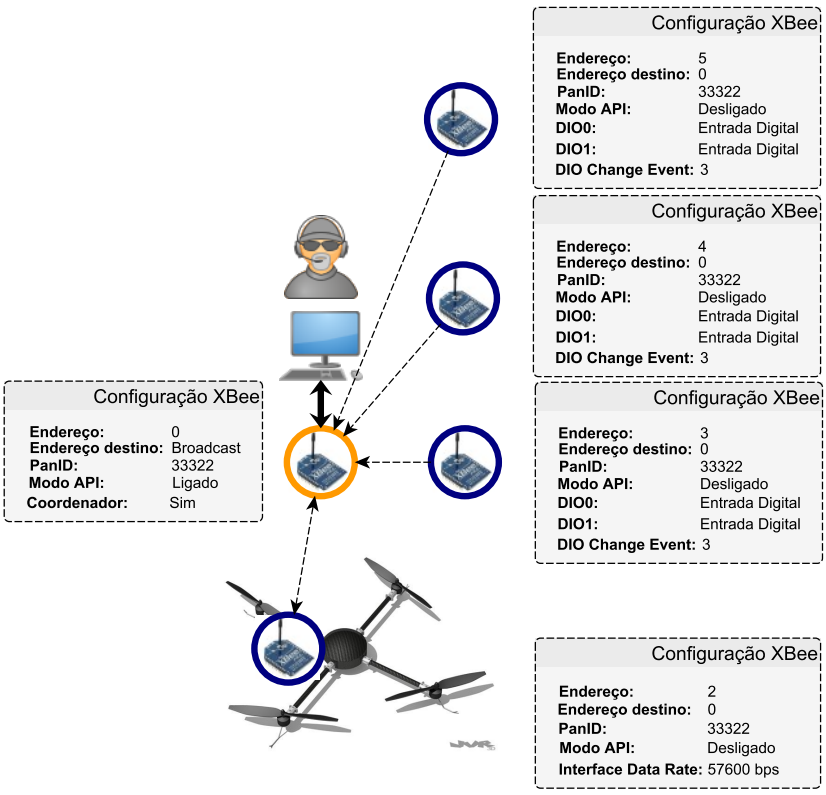

Figura 5.18: Configuração da rede sem fio

Para realizar uma demonstração, uma rede estrela foi montada com as configurações de acordo com a figura 5.18.

Um arquivo de descrição de missão hipotética aplicada a essa configuração de rede foi criado com a ajuda de um software destinado à edição de novas missões criado como parte de outro trabalho de conclusão[11].

A missão criada especifica três nodos estáticos posicionados em três pontos distintos nos limites do parque farroupilha em Porto Alegre. Um localizado no campo de futebol, outro entre o chafariz principal do parque e o lago e o último entre o auditório Araújo e o campo de futebol. Todos os nodos como sensores de presença. Dois pontos de interesse foram adicionados sobre o lago, de modo ao nós móveis monitorarem a movimentação ao seu redor.

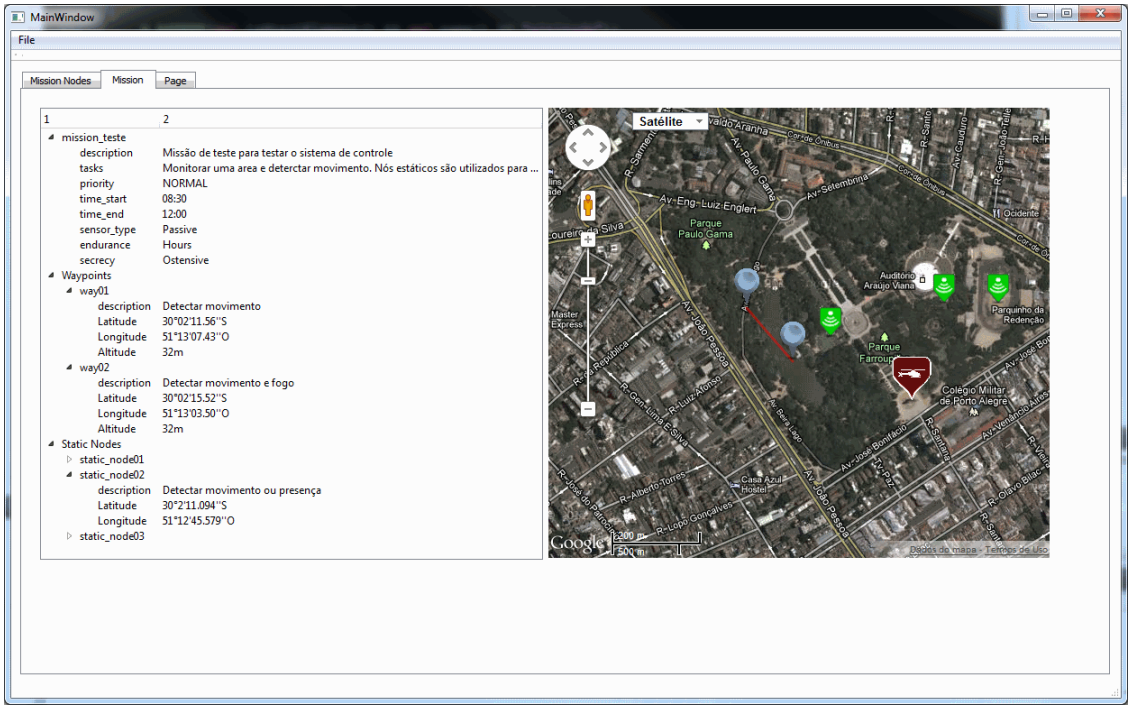

Figura 5.19: Ambiente de missão

A figura 5.19 mostra o ambiente de missão criado após a MDL ser lida e analisada, assim como todos os nodos estáticos criados (figura 5.20).

Em seguida, as adições de nós móveis devem ser feitas. Para esse caso, um quadricóptero foi inserido à rede (figura 5.21) e posicionado inicialmente ao lado do arco do parque da redenção.

A configuração de cada nó através da definição de cada endereço no software da estação de solo, deve ser compatível com a configuração do endereço de cada módulo XBee físico.

Para iniciar a ação do nodo móvel, os *waypoints* da missão devem ser enviados ao nó móvel. O botão *"Assign Waypoints"* relaciona os pontos de interesse inseridos na missão e envia ao primeiro quadricóptero disponível. O nó é redirecionada com isso, aos marcadores indicados em azul no mapa da fgura 5.19.

A figura 5.22 mostra os *waypoints* enviados pela estação de solo lidos a partir do software *Kopter Tool* da própria desenvolvedora do quadricóptero. Mostra-se assim, que a execução de comandos ao helicóptero pela estação de solo é possível, da mesma maneira que é feita através do software oficial de telemetria e comando da *Mikrokopter*.

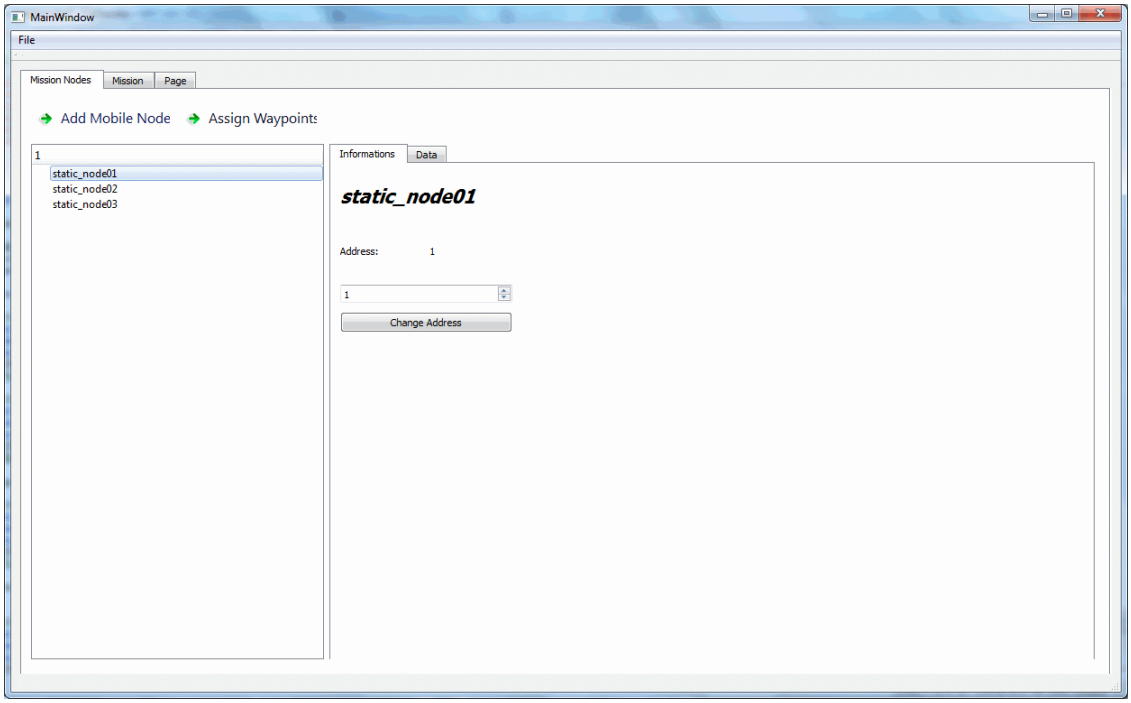

Figura 5.20: Nós estáticos da missão

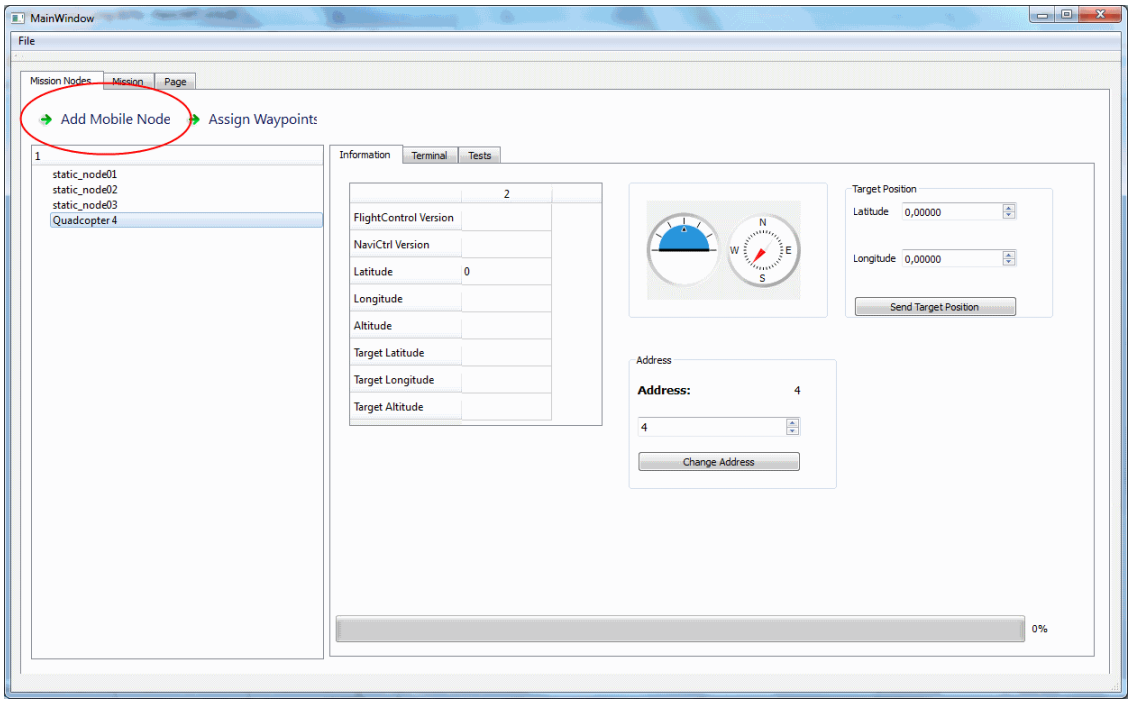

Figura 5.21: Adiçao do nó móvel da missão

Pressionando o primeiro botão de uma das placas de desenvolvimento dos módulos XBees, uma mensagem é enviada ao coordenador, que a interpreta como um evento da rede de sensores sem fio. A partir dessa mensagem, os pontos de interesses atribuídos ao nó móvel sao apagados, e a coordenada geográfica do sensor estático que emitiu o evento é enviado como um *waypoint* ao quadricóptero.

Como exemplo, o nó estático com endereço quatro envia uma mensagem de botão

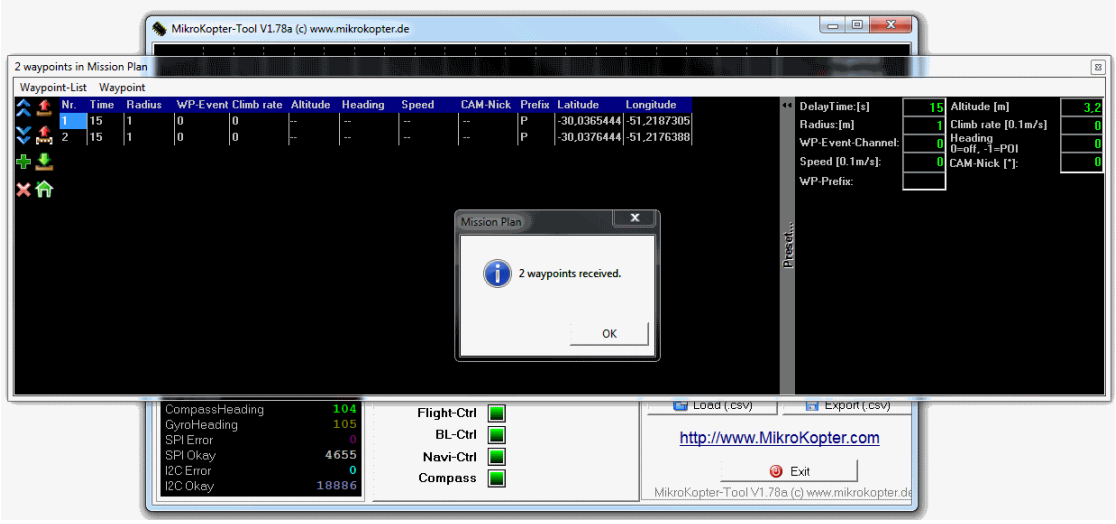

Figura 5.22: Waypoints gravados no quadricóptero no início da missão

pressionado ao nó coordenador. Sua posição geográfica é então definida como único ponto de interesse do helicóptero. Para verificar, mais uma vez foi usado o software *Kopter Tool*. A figura 5.23 mostra exatamente a coordenada do nó sensor.

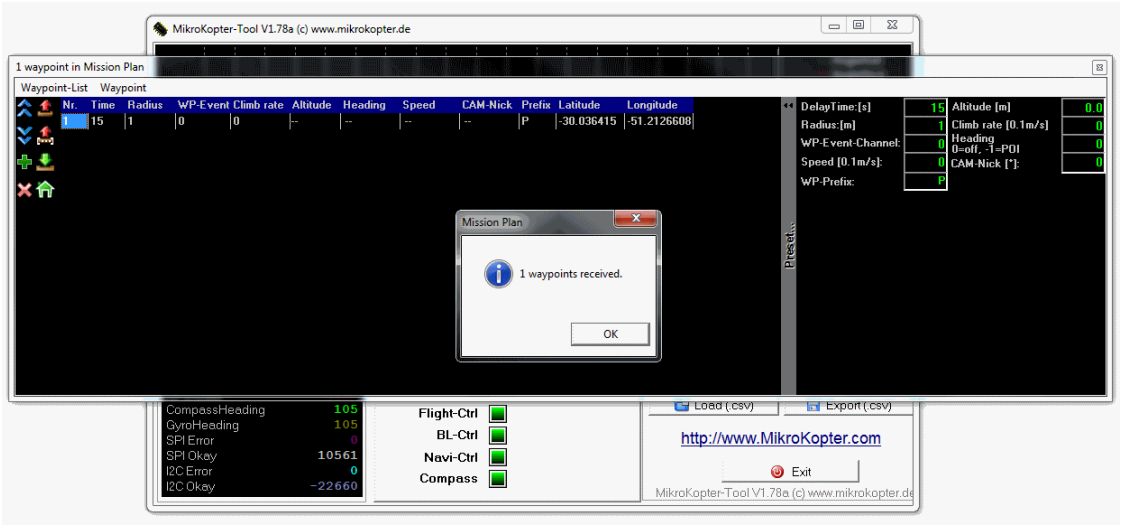

Figura 5.23: Waypoint gravado no quadricóptero após um evento ser gerado por um nó sensor estático

5.4 Trabalhos Relacionados

Uma série de comunidades relacionadas a aeromodelismo desenvolve software de telemetria para quadricópteros dos mais variados fabricantes. *ArduCopter* e *OpenPilot* são alguns exemplos de helicópteros não tripulados que possuem estações de solo para aquisição de seus dados. A própria *Mikrokopter* possui um software para configuração e monitoramento de seus VANTs chamado *Kopter Tool*. Entretanto, o objetivo de todos é a análise de dados para apenas uma aeronave já que seus produtos não possuem foco inicial de interagirem entre si.

Estações base para centralização de informações também é proposto em [13]. Também possui requisitos de modularidade para permitir o interfaceamento com os mais diversos protocolos de comunicação. Porém tem a característica de ser embarcado, e dessa forma as restrições de hardware devem ser levadas em conta.

Uma plataforma semelhante a esse trabalho é proposta em [19]. O projeto prevê funcionalidades com o objetivo de cooperação entre VANTs, sensores e atuadores em solo, e nodos móveis acoplados em veículos em solo ou até mesmo em pessoas.

6 CONCLUSÃO

Foi possível nesse trabalho a implementação de diferentes interfaces com nodos em uma rede de sensores sem fio. A comunicação com um VANT, representando um nó móvel, foi implementada parcialmente de forma a monitorar os seus movimentos, tanto pela aquisição dos dados de seus sensores inerciais embarcados como pelas coordenadas geográficas adquiridas pelo seu GPS, e, além disso, controlar a sua trajetória através do envio de pontos de interesse.

A estrutura do software implementada permitiu que o propósito inicial, da adição de diversos nodos distintos a rede, fosse atingido. Assim, uma rede estrela de nós sensores heterogêneos foi criada de modo a cumprir determinada missão descrita pela MDL. Além disso, a estrutura permite que novos nodos e novas interfaces possam ser incluídas de maneira simples e intuitiva para futuros desenvolvedores.

Foi possível demonstrar a centralização de informações de todos o nós da rede em apenas uma interface de usuário. Possibilitando o controle da rede através do conhecimento situacional do ambiente em que a missão estava inserida.

Em comparação com uma rede estrela, uma rede *mesh* possui uma abrangência muito maior, fazendo com que uma região muito maior possa ser monitorada. Entretanto a sua construção passa a ser mais complexa, já que algoritmos específicos para redirecionamento de mensagens precisa ser implementado nos próprios nodos da rede. Em contrapartida, a adição de inteligência aos nós estáticos pode também inserir certos tipos de cooperação entre eles, de forma que um conjunto de sensores possa cumprir determinada missão.

Com a nova topologia, mais de uma estação de solo poderia ser inserida em diversos pontos da rede pra realizar telemetrias mais específicas de cada região coberta pela rede.

Outra frente de trabalho é o incremento da estação de solo. A telemetria para o quadricóptero pode ser ampliada cobrindo todos os recursos do VANT, assim como o suporte para diferentes tipos de sensores estáticos, como sensores de temperatura, por exemplo, criando um *widget* para sensores analógicos.

Novas interfaces poderiam ser criadas para que a abrangência de usuário que poderiam usufruir das informações fosse maior. Em [17], é proposto uma solução de centralização de informações de uma rede sem fio utilizando dispositivos móveis com sistema operacional Android. Para isso além do software de interface com o usuário presente nos celulares, foi necessária a presença de um nodo servidor que além de centralizar as informações presentes na rede, é capaz de enviar para os dispositivos apenas os dados relevantes ao usuário. Algo semelhante poderia ser proposto, fazendo com que a estação de solo, além de monitorar e controlar uma rede de sensores sem fio, assumisse um papel de servidor remoto de informações para diferentes plataformas.

REFERÊNCIAS

[1] *IEEE 802.15.4f Amendment 2: Active Radio Frequency Identification (RFID) System Physical Layer (PHY)*, 2012.

[2] I.F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. Wireless sensor networks: a survey. *Computer Networks*, pages 393–421, 2002.

[3] Majid Bahrepour, Nirvana Meratnia, and Paul J. M. Havinga. Sensor fusion-based event detection in wireless sensor networks. *IEEE Mobile and Ubiquitous Systems: Networking & Services*, 2009.

[4] Nick Baker. Zigbee and bluetooh: Strengths and weaknesses for industrial application. *IEE Computing & Control Engineering*, 2005.

[5] Ed Callaway, Paul Gorday, Lance Hester, Jose A. Gutierrez, Marco Naeve, Bob Heile, and Venkat Bahl. Home networking with IEEE 802.15.4: A developing standard for low-rate wireless personal area networks. *IEEE Communications Magazine*, pages 70–77, 2002.

[6] Krishnendu Chakrabarty, S. Sitharama Iyengar, Hairong Qi, and Eungchun Cho. Grid coverage for surveillance and target location in distributed sensor networks. *IEEE Transactions on Computers*, 2002.

[7] W. Chebbi, M. Benjemaa, A. Kamoun, M. Jabloun, and A. Sahli. Development of a wsn integrated weather station node for an irrigation alert program under tunisian conditions. *8th International Multi-Conference on Systems, Signals & Devices*, 2011.

[8] Edison Pignaton de Freitas. *Cooperative Context-Aware Setup and Performance of Surveillance Missions Using Static and Mobile Wireless Sensor Networks*. PhD thesis, 2011.

[9] Edison Pignaton de Freitas, Tales Heimfarth, Carlos Eduardo Pereira, Armando Morado Ferreira, Flavio Rech Wagner, and Tony Larsson. Multi-agent support in a middleware for mission-driven heterogeneous sensor networks. *The Computer Journal*, 2009.

[10] Chien-Liang Fok, Gruia-Catalin Roman, and Chenyang Lu. Rapid development and flexible deployment of adaptive wireless sensor network applications. *25th IEEE International Conference on Distributed Computing Systems*, 2005.

[11] Athos Alexandre Lima Fontanari, Flavio Rech, and Carlos Eduardo. Sistema de planejamento e controle de missao de um veiculo aereo nao-tripulado. 2011.

[12] Jose A. Gutierrez, Marco Naeve, Ed Callaway, Monique Bourgeois, Vinay Milter, and Bob Heile. IEEE 802.75.4: A developing standard for low-power low-cost wireless personal area networks. *IEEE Network*, 2001.

[13] Jerry Daniel J, Senju Thomas Panicker, Lijo Thomas, Jacob T. Mathew, and Ann Mathew. Industrial grade wireless base station for wireless sensor networks. *Electronics Computer Technology (ICECT) International Conference*, 2011.

[14] Jae-Joon Lee, Bhaskar Krishnamachari, and C.-C. Jay Kuo. Impact of heterogeneous deployment on lifetime sensing coverage in sensor networks. *IEEE SECON*, pages 367–376, 2004.

[15] Samuel R. Madden, Michael J. Franklin, Joseph M. Hellerstein, and Wei Hong. Tinydb: An acquisitional query processing system for sensor networks. *ACM Transactions on Database Systems*, 2005.

[16] MikroKopter: modular multicopter systems. http://www.mikrokopter.de, 2008.

[17] Nuno Moreira, Marco Venda, Catarina Silva, Luis Marcelino, and Antonio Pereira. @sensor - mobile application to monitor a wsn. 2011.

[18] Ivan Muller, Carlos E. Pereira, Joao C. Netto, Rodrigo Allgayer, and Rodolfo Dresh. Rede de sensores sem fio aplicada no monitoramento de bancos de baterias para nobreaks. *XVIII Congresso Brasileiro de Automatica*, 2010.

[19] Anibal Ollero, Markus Bernard, Marco La Civita, Lodewijk van Hoesel, Pedro J. Marron, Jason Lepley, and Eduardo de Andres. Aware: Platform for autonomous self-deploying and operation of wireless sensor-actuator networks cooperating with unmanned aerial vehicles. *IEEE International Workshop on Safety, Security and Rescue Robotics*, 2007.

[20] Carlos Eduardo Pereira. Integracao de redes de sensores sem fio com veiculos aereos nao-tripulados (vants). *Premio Santander 2010*, 2010.

[21] Skydrones. http://www.skydrones.com.br.

[22] Haitao Xiao, Yixuan Gong, Harutoshi Ogai, Jie Zhang, Xiaohong Zou, Takenari Otawa, and Takunori Tsuji. A data collection system in wireless network integrated wsn and zigbee for bridge health diagnosis. *SICE Annual Conference 2011*, 2011.