Modelo de arquitetura em camadas para interconexão de sistemas em SANT

Emerson Alberto Marconato

SERVIÇO DE PÓS-GRADUAÇÃO DO ICMC-USP

Data de Depósito:

Assinatura: _____

Emerson Alberto Marconato

Modelo de arquitetura em camadas para interconexão de sistemas em SANT

Tese apresentada ao Instituto de Ciências Matemáticas e de Computação – ICMC-USP, como parte dos requisitos para obtenção do título de Doutor em Ciências – Ciências de Computação e Matemática Computacional. *VERSÃO REVISADA*

Área de Concentração: Ciências de Computação e Matemática Computacional

Orientadora: Profa. Dra. Kalinka Regina Lucas Jaquie Castelo Branco

Coorientador: Prof. Dr. Denis Fernando Wolf

USP – São Carlos
Janeiro de 2017

Ficha catalográfica elaborada pela Biblioteca Prof. Achille Bassi
e Seção Técnica de Informática, ICMC/USP,
com os dados fornecidos pelo(a) autor(a)

M321m Marconato, Emerson Alberto
 Modelo de arquitetura em camadas para
interconexão de sistemas em SANT / Emerson
Alberto Marconato; orientadora Kalinka Regina
Lucas Jaquie Castelo Branco; coorientador Denis
Fernando Wolf. - São Carlos - SP, 2017.
 237 p.

 Tese (Doutorado - Programa de Pós-Graduação em
Ciências de Computação e Matemática Computacional)
- Instituto de Ciências Matemáticas e de Computação,
Universidade de São Paulo, 2017.

 1. Modelo de Arquitetura. 2. VANT. 3. SANT.
4. Modelo de Referência. 5. Sistemas Embarcados
Críticos. I. Branco, Kalinka Regina Lucas Jaquie
Castelo, orient. II. Wolf, Denis Fernando, coorient.
III. Título.

Emerson Alberto Marconato

Layered architecture model for interconnection of systems in UAS

Doctoral dissertation submitted to the Instituto de Ciências Matemáticas e de Computação – ICMC-USP, in partial fulfillment of the requirements for the degree of the Doctorate Program in Computer Science and Computational Mathematics. *FINAL VERSION*

Concentration Area: Computer Science and Computational Mathematics

Advisor: Profa. Dra. Kalinka Regina Lucas Jaquie Castelo Branco

Co-advisor: Prof. Dr. Denis Fernando Wolf

USP – São Carlos
January 2017

*Este trabalho é dedicado aos que amam o ofício de ensinar ofícios.
E em especial, à minha amada esposa Hieda e meu amado filho Caio.*

AGRADECIMENTOS

Agradeço a Deus pelo dom da vida, possibilitando trilhar e concluir esta etapa de minha carreira.

Aos meus pais, Antônio e Neide, que me deram amor incondicional, educação e que não mediram esforços para que eu pudesse me graduar e alçar voos com minhas próprias asas.

À minha amada esposa, amiga e companheira Hieda, que com seu amor, paciência e carinho, ajudou-me superar esta etapa, sempre com palavras de apoio e incentivo e acima de tudo, teve compreensão nos momentos em que estive ausente.

À meu amado filho Caio, pelos momentos ausentes, jogos de futebol e de vídeo game que não participei.

À minha família: Wiliam, Marlene e Mariane; Regiane, Olacir, Ana Julia, Isabela e Lívia; Glauciane e Valter; Odilon e Consolação; Vander, Engrid, Fernanda e Felipe, pelo incentivo e carinho.

À minha estimada orientadora Kalinka, pela dedicação, orientação, horas dispendidas, apoio, ideias, críticas sempre construtivas e principalmente pelas palavras sempre amigas e animadoras nos momentos necessários.

Ao amigos e colegas do LSEC, Adimara, Arthur, Daniel, Douglas, Lucas, Mariana, Natássya, Querino, Rayner e Rodolfo, pelo companheirismo e troca de experiências que certamente enriqueceram este trabalho.

Ao meu amigo Paulinho, por quem tenho grande apresso e amizade e que nunca deixou de me apoiar.

Aos meus alunos, cuja "fome" de conhecimento é o meu alimento para busca de novos horizontes.

Aos órgãos de fomento a pesquisa, FAPESP e INCT, cujo apoio em forma de aquisição de equipamentos e infraestrutura para o LSEC auxiliaram na execução desta pesquisa.

À USP, em especial aos funcionários e professores do ICMC.

Agradecimentos à Combr, Famema, Senac e Univem, empresas e entidades acadêmicas que de uma forma ou outra participaram desta empreitada, em especial aos colegas que ali trabalham e sempre proferiram palavras de apoio.

Por fim a todos professores, meus Mestres, que desde os idos de 1977 participaram de

minha formação.

*“Inventar é imaginar o que ninguém pensou;
é acreditar no que ninguém jurou;
é arriscar o que ninguém ousou;
é realizar o que ninguém tentou.
Inventar é transcender.”
(Alberto Santos Dumont)*

RESUMO

MARCONATO, E. A.. **Modelo de arquitetura em camadas para interconexão de sistemas em SANT**. 2017. 237 f. Tese (Doutorado em Ciências – Ciências de Computação e Matemática Computacional) – Instituto de Ciências Matemáticas e de Computação (ICMC/USP), São Carlos – SP.

Modelos de arquitetura têm sido utilizados para permitir o desenvolvimento mais adequado e estruturado de sistemas, desde os mais simples até os mais complexos. A utilização desses modelos em sistemas embarcados, principalmente quando se trata de sistemas embarcados críticos, como é o caso de veículos aéreos não tripulados (VANT), visam a permitir conformidades de padrões, redução no tempo de produção, redução e facilidade no processo de manutenção e desenvolvimento. Sistemas embarcados críticos possuem requisitos específicos, tais como alta confiabilidade e resposta em tempo real, segurança e desempenho. A definição de um modelo arquitetural que permita que esses quesitos sejam levados em consideração, propicie o atendimento aos padrões, além de permitir o desenvolvimento correto e acelerado é inovador, permitindo que não só a comunidade científica venha a ter benefícios com a sua concepção, mas também a indústria brasileira possa ganhar. Nesse sentido, este trabalho desenvolveu um modelo de arquitetura para a interconexão de sistemas aéreos não tripulados (SANTs) em *Unified Modeling Language* (UML)/*System Modelling Language* (SysML) denominado LARISSA (*Layered ARchitecture model for Interconnection of SystemS in uAs*). Como resultado deste trabalho foi possível a modelagem e especificação completa de um SANT fazendo uso desse modelo e a realização de diversos experimentos que permitiram validar o LARISSA. Os experimentos, focados na parte de comunicação, permitiram a concepção de um simulador de redes de VANTs. Os resultados obtidos demonstraram a eficiência e a eficácia do modelo de arquitetura LARISSA, além da sua flexibilidade em permitir que diferentes experimentos possam ser realizados, o que auxilia na obtenção de dados que facilitam o processo de certificação desses VANTs.

Palavras-chave: Modelo de Arquitetura, VANT, SANT, Modelo de Referência, Sistemas Embarcados Críticos.

ABSTRACT

MARCONATO, E. A.. **Modelo de arquitetura em camadas para interconexão de sistemas em SANT**. 2017. 237 f. Tese (Doutorado em Ciências – Ciências de Computação e Matemática Computacional) – Instituto de Ciências Matemáticas e de Computação (ICMC/USP), São Carlos – SP.

Architecture models have been used to allow more adequate and structured development of systems, from the simplest to the most complex. The use of models in embedded systems are used to enable compliance standards, reduction in production time and ease of maintenance and development, especially when it comes to critical embedded systems such as unmanned aerial vehicles. Critical embedded systems have specific requirements, such as high reliability and real-time response, security and performance. The definition of an architectural model that deals with these requirements and standards and provides the service to standards allowing the correct and fast development is innovative and will provide benefits to the scientific and industrial communities. In this sense, this work developed an architectural model for the interconnection of Unmanned Aerial Systems (UAS) in Unified Modeling Language (UML)/System Modelling Language (SysML) called LARISSA (**L**ayered **A**Rchitecture model for **I**nterconnection of **S**ystem**S** in **u**A**s**). As a result of this development it was possible the modelling and specification of a complete UAS and the conduction of several experiments that allowed LARISSA validation. The experiments focused on the communication enabled the design of a UAV networks simulator. The results showed the efficiency and effectiveness of LARISSA architecture model, as well as its flexibility in carrying out different experiments, which helps in getting data to facilitate the certification process of these UAVs.

Key-words: Architecture Model, UAV, UAS, Reference Model, Critical Embedded Systems.

LISTA DE ILUSTRAÇÕES

Figura 1 – Desafios de integração dos SANTs no espaço aéreo: Oportunidades e Necessidades.	37
Figura 2 – Exemplos de Tipos de VANTs	47
Figura 3 – VANT e seu sistema de suporte	50
Figura 4 – VANTs de interesse para MCAP III	58
Figura 5 – Diagrama de contexto do UCAP	59
Figura 6 – Arquitetura MCAP III/UCAP	59
Figura 7 – Principais componentes de um sistema VANT	61
Figura 8 – Visão Geral da arquitetura do Computador de Controle de Missão	62
Figura 9 – Arquitetura de comunicação entre o VANT e estação de controle terrestre	63
Figura 10 – Arquitetura RFCSA	65
Figura 11 – SOA embarcada no RFCSA	66
Figura 12 – Arquitetura modular em camadas	67
Figura 13 – Exemplo de arquitetura Federada e IMA	69
Figura 14 – <i>Avionic System Architecture</i>	70
Figura 15 – Arquitetura ARINC 653 baseada em Linux	72
Figura 16 – LARISSA	78
Figura 17 – Camada Abstração do Sistema	79
Figura 18 – Camada Física	80
Figura 19 – Camada Física - Subcamada de Estrutura	81
Figura 20 – Camada Física - Subcamada Aviônica	83
Figura 21 – Camada Física - Subcamada de Energia	84
Figura 22 – Camada Física - Subcamada de Sistemas Auxiliares	86
Figura 23 – Camada RTOS Distribuído	87
Figura 24 – Camada Monitoramento e Controle	88
Figura 25 – Camada Navegação e Serviços	89
Figura 26 – Camada Missão	89
Figura 27 – Camada Física	90
Figura 28 – Camada Estação de Controle Terrestre	91
Figura 29 – LARISSA Use Case View	94
Figura 30 – LARISSA Logical View	94
Figura 31 – LARISSA Process View	95
Figura 32 – LARISSA Implementation View	96

Figura 33 – LARISSA Deployment View	96
Figura 34 – <i>Flight Sensor</i>	97
Figura 35 – <i>Profile</i> LARISSA	98
Figura 36 – Fluxo <i>profile</i> LARISSA e <i>larissamodel2simulation</i>	106
Figura 37 – Exemplo de arquivo NED - modo <i>Design</i>	112
Figura 38 – Network NED - modo <i>Design</i>	113
Figura 39 – <i>Use Case View</i> dos cenários	118
Figura 40 – <i>Logical View</i> dos cenários	119
Figura 41 – <i>Process View</i> dos cenários	119
Figura 42 – Aplicando o <i>profile</i>	120
Figura 43 – Atribuição de Valores a <i>tagged value</i>	120
Figura 44 – <i>Implementation View</i> dos cenários	121
Figura 45 – <i>Deployment View</i> dos cenários	121
Figura 46 – Uso do <i>larissamodel2simulation</i>	122
Figura 47 – Pacotes Transmitidos (Tx) e Recebidos (Rx) das Simulações do Cenário I	125
Figura 48 – Consumo de bateria do Cenário I	126
Figura 49 – UAV[1] sem comunicação direta com GCS	128
Figura 50 – UAV[0] roteando pacote do UAV[1]	128
Figura 51 – Pacotes Transmitidos (Tx) e Recebidos (Rx) das Simulações do Cenário II	129
Figura 52 – Consumo Bateria do Cenário II	130
Figura 53 – UAV[0] sem comunicação direta com GCS	131
Figura 54 – UAV[2] roteando pacote do UAV[0]	132
Figura 55 – Pacotes Transmitidos (Tx) e Recebidos (Rx) das Simulações do Cenário III	132
Figura 56 – <i>Delay</i> fim-a-fim das simulações do Cenário III	133
Figura 57 – Comparação dos pacotes transmitidos com sucesso pelo IEEE 802.11n.	136
Figura 58 – Comparação de pacotes transmitidos com sucesso pelo IEEE 802.15.4.	136
Figura 59 – Comparação de pacotes transmitidos com sucesso em configuração estrela	137
Figura 60 – Comparação de pacotes transmitidos com sucesso em configuração <i>mesh</i>	137
Figura 61 – Análise do atraso fim a fim do IEEE 802.11n - Topologia estrela	138
Figura 62 – Análise do atraso fim a fim do IEEE 802.11n - Topologia <i>mesh</i>	138
Figura 63 – Análise do atraso fim a fim no IEEE 802.15.4 - Topologia estrela	139
Figura 64 – Análise do atraso fim a fim no IEEE 802.15.4 - Topologia <i>Mesh</i>	139
Figura 65 – Arquitetura - SLUGS	142
Figura 66 – <i>Use Case View</i> - SLUGS	143
Figura 67 – <i>Logical View</i> - SLUGS	143
Figura 68 – <i>Process View</i> - SLUGS dos sensores	144
Figura 69 – <i>Process View</i> - SLUGS da controladora	144
Figura 70 – O Pacote <i>Flight Sensors</i> - SLUGS	145
Figura 71 – O Pacote <i>Communication Device</i> - SLUGS	145

Figura 72 – O Pacote <i>Controller Board</i> - <i>SLUGS</i>	146
Figura 73 – <i>Implementation View</i> - <i>SLUGS</i>	147
Figura 74 – <i>Deployment View</i> - <i>SLUGS</i>	147
Figura 75 – <i>SLUGS</i> - Esquema Controladora	148
Figura 76 – <i>SLUGS</i> - LSEC	148
Figura 77 – <i>SLUGS</i> - HILS	149
Figura 78 – Arquitetura - <i>YAPA</i>	150
Figura 79 – <i>Use Case View</i> - <i>YAPA</i>	151
Figura 80 – <i>Logical View</i> - <i>YAPA</i>	151
Figura 81 – <i>Process View</i> - <i>YAPA</i> dos sensores - parte 1	152
Figura 82 – <i>Process View</i> - <i>YAPA</i> dos sensores - parte 2	152
Figura 83 – <i>Process View</i> - <i>YAPA</i> da controladora	153
Figura 84 – O Pacote <i>Flight Sensors</i> - <i>YAPA</i>	153
Figura 85 – O Pacote <i>Communication Device</i> - <i>YAPA</i>	154
Figura 86 – O Pacote <i>Controller Board</i> - <i>YAPA</i>	154
Figura 87 – <i>Implementation View</i> - <i>YAPA</i>	155
Figura 88 – <i>Deployment View</i> - <i>YAPA</i>	156
Figura 89 – <i>YAPA</i> - LSEC	156
Figura 90 – <i>YAPA</i> - Esquema	157
Figura 91 – <i>AVALON</i> - <i>Design</i>	158
Figura 92 – <i>Use Case View</i> - <i>AVALON</i>	158
Figura 93 – <i>Logical View</i> - <i>AVALON</i>	159
Figura 94 – <i>Process View</i> - <i>AVALON</i> do sistema de propulsão	160
Figura 95 – <i>Process View</i> - <i>AVALON</i> da controladora	160
Figura 96 – <i>Process View</i> - <i>AVALON</i> da aeronave	161
Figura 97 – O Pacote <i>PropulsionSystem</i> - <i>AVALON</i>	161
Figura 98 – O Pacote <i>Controller Board</i> - <i>AVALON</i>	162
Figura 99 – O Pacote <i>Aircraft</i> - <i>AVALON</i>	162
Figura 100 – <i>Implementation View</i> - <i>AVALON</i>	163
Figura 101 – <i>Deployment View</i> - <i>AVALON</i>	164
Figura 102 – Estrutura do AVENS - integração entre os simuladores	166
Figura 103 – Pacote <i>ControllerBoard</i> com o <i>profile Controller</i>	167
Figura 104 – Dispositivo de <i>RadioModem</i>	168
Figura 105 – Diagrama de blocos internos - VANTs	168
Figura 106 – Diagrama de classes <i>Arbitrary Mobility</i>	169
Figura 107 – Modelagem da aeronave no <i>plugin</i> do X-Plane	170
Figura 108 – Fluxo de comunicação entre OMNeT++ e X-Plane.	170
Figura 109 – Representação da trajetória de navegação - <i>North-East-Up</i>	172
Figura 110 – Resultados da simulação dos cenários com 3 VANTs voando em cooperação	173

Figura 111 – Resultados das simulações dos cenários com 5 VANTs voando em cooperação 173

LISTA DE QUADROS

Quadro 1 – Classificação RPAS/RPA - proposta Reino Unido.	44
Quadro 2 – Classificação RPAS/RPA - proposta ANAC	45
Quadro 3 – Classificação RPAS/RPA - proposta Europeia	45
Quadro 4 – Classificação conforme propósito de utilização e raio de atuação.	46
Quadro 5 – Requisitos atendidos pela MCAP.	60
Quadro 6 – Requisitos atendidos pela MCC.	64
Quadro 7 – Requisitos atendidos pela RFCSA.	65
Quadro 8 – Requisitos atendidos pela LMA.	67
Quadro 9 – Comparação entre arquitetura federada e IMA	69
Quadro 10 – Requisitos atendidos pelo Mini VTOL.	71
Quadro 11 – Requisitos atendidos pelo projeto ANKA	71
Quadro 12 – Requisitos atendidos pelo ARINC	73
Quadro 13 – Comparativo entre os trabalhos encontrados na literatura aberta	74
Quadro 14 – Alto nível de abstração do <i>profile</i> LARISSA	98
Quadro 15 – <i>SystemAbstraction Profile</i>	99
Quadro 16 – <i>Aerial Profile</i>	99
Quadro 17 – <i>Physical Profile</i>	99
Quadro 18 – <i>Structure Profile</i>	100
Quadro 19 – <i>AvionicsElectronics Profile</i>	100
Quadro 20 – <i>Sensor Profile</i>	100
Quadro 21 – <i>EngineSensor Profile</i>	100
Quadro 22 – <i>FlightSensor Profile</i>	101
Quadro 23 – <i>Ground Profile</i>	101
Quadro 24 – <i>Physical Profile</i>	102
Quadro 25 – <i>GroundStation Profile</i>	102
Quadro 26 – Principais características do VANT DT-18	134
Quadro 27 – Diferentes tipos de dados trocados durante o voo	134
Quadro 28 – Resultados do experimento de uma FANET	135
Quadro 29 – Artigos aceitos na revisão sistemática	205
Quadro 30 – <i>AirTrafficControl Profile</i>	207
Quadro 31 – <i>WeatherMonitoringForecasting Profile</i>	207
Quadro 32 – <i>ServoAndActuator Profile</i>	208
Quadro 33 – <i>Infrastructure Profile</i>	208

Quadro 34 – <i>Antenna Profile</i>	208
Quadro 35 – <i>Communications Profile</i>	209
Quadro 36 – <i>AirToGround Profile</i>	209
Quadro 37 – <i>TelecommandTelemetry Profile</i>	209
Quadro 38 – <i>DataRelay Profile</i>	210
Quadro 39 – <i>AirToAir Profile</i>	210
Quadro 40 – <i>Controller Profile</i>	211
Quadro 41 – <i>Energy Profile</i>	211
Quadro 42 – <i>Propulsion Profile</i>	211
Quadro 43 – <i>OnBoardPowerGenaration Profile</i>	212
Quadro 44 – <i>PowerSupplyAndConditioning Profile</i>	212
Quadro 45 – <i>Fuel Profile</i>	212
Quadro 46 – <i>Electricity Profile</i>	213
Quadro 47 – <i>AuxiliarySystems Profile</i>	213
Quadro 48 – <i>DistributedRTOS Profile</i>	213
Quadro 49 – <i>MonitoringAndControl Profile</i>	214
Quadro 50 – <i>NavigationAndServices Profile</i>	214
Quadro 51 – <i>Mission Profile</i>	215

LISTA DE CÓDIGOS-FONTE

Código-fonte 1	– Trecho de código do <i>larissamodel2simulation</i>	103
Código-fonte 2	– Exemplo de arquivo NED - modo <i>Source</i>	108
Código-fonte 3	– Network NED - modo <i>Source</i>	111
Código-fonte 4	– Parâmetros gerados pelo <i>larissamodel2simulation</i>	114
Código-fonte 5	– <i>Activator.java</i>	217
Código-fonte 6	– <i>Generate.java</i>	218
Código-fonte 7	– <i>generate.mtl</i>	230
Código-fonte 8	– <i>UMLservices.java</i>	235
Código-fonte 9	– <i>umlservices.mtl</i>	236

LISTA DE TABELAS

Tabela 1 – Cenários Avaliados	123
Tabela 2 – Parâmetros e valores das simulações do Cenário I	124
Tabela 3 – Percentual de perda de pacotes - Cenário I	125
Tabela 4 – Parâmetros e valores das simulações do Cenário II	127
Tabela 5 – Percentual de perda de Pacotes - Cenário II	128
Tabela 6 – Parâmetros e valores das Simulações do Cenário III	130
Tabela 7 – Percentual de perda de pacotes - Cenário III	131
Tabela 8 – Valores dos parâmetros para os cenários de simulação implementados no AVENS	171
Tabela 9 – Total de artigos - primeiro levantamento	203
Tabela 10 – Distribuição por ano dos artigos aceitos na revisão sistemática	204
Tabela 11 – Requisitos atendidos	204

LISTA DE ABREVIATURAS E SIGLAS

ADS-B	...	<i>Automatic Dependent Surveillance-Broadcast</i>
ANAC		<i>Agência Nacional de Aviação Civil</i>
AODV	...	<i>Ad hoc On-Demand Distance Vector</i>
APIs		<i>Application Programming Interfaces</i>
APU		<i>Auxiliary Power Unit</i>
AVALON	.	<i>Autonomous Vertical takeOff and laNding</i>
AVENS	...	<i>Aerial Vehicle Network Simulator</i>
C4ISR		<i>Command, Control, Communications, Computers, Intelligence, Surveillance and Reconnaissance</i>
CAA		<i>Civil Aviation Authority</i>
CAN		<i>Ethernet Controller Area Network</i>
CHT		<i>Cylinder Head Temperature</i>
COTS		<i>Commercial off-the-shelf</i>
CPDLC	..	<i>Controller–Pilot Data Link Communications</i>
CPUs		<i>Central Processing Units</i>
DDD		<i>Dull, Dirty, Dangerous</i>
DoD		<i>Department of Defense</i>
DoDAF	...	<i>Department of Defense Architecture Framework</i>
DSC		<i>Digital Signal Controllers</i>
DTM		<i>Data Transfer Mechanism</i>
E/S		<i>Entrada/Saída</i>
EASA		<i>European Aviation Safety Agency</i>
EGT		<i>Exhaust Gas Temperature</i>
ENAC		<i>Ecole Nationale de l’Aviation Civile</i>
ERMP		<i>Extended Range/Multi-Propose</i>
ERSG		<i>European RPAS Steering Group</i>
EUA		<i>Estados Unidos da América</i>
FAA		<i>Federal Aviation Administration</i>
FANETs	..	<i>Flying Ad Hoc Network</i>
FCS		<i>Future Combat Systems</i>
GbE		<i>Gigabit Ethernet</i>

GCS *Ground Control Station*
 GEO *Geostationary Orbit*
 GPIO *General Purpose Input/Output*
 GPS *Global Positioning System*
 HALE *High altitude long endurance*
 HILS *Hardware in the Loop Simulation*
 I/O *Input/Output*
 I2C *Inter-Integrated Circuit*
 IDE *Integrated Development Environment*
 IMA *Integrated Modular Avionics*
 IMU *Inertial Measurement Unit*
 INCT-SEC Instituto Nacional de Ciência e Tecnologia em Sistemas Embarcados Críticos
 IP *Internet Protocol*
 IPC *Inter-Process Communication*
 ISM *Industrial, Scientific and Medical*
 ISO *International Organization for Standardization*
 LAN *Local Area Network*
 LARISSA *Layered ARchitecture model for Interconnection of SystemS in uAs*
 LEO *Low Earth Orbit*
 Li-Po *Lithium Polymer*
 LOS *line-of-sight*
 LRMs *Line Replaceable Modules*
 LRUs *Line Replaceable Units*
 LSEC Laboratório de Sistemas Embarcados Críticos
 MALE ... *Medium altitude long endurance*
 MANETs . *Mobile Ad Hoc Networks*
 MARTE .. *Modeling And Analysis Of Real-Time Embedded Systems*
 MAV *Micro UAV*
 MCAP ... *Manned/Unmanned Common Architecture Program*
 MCC *Mission Control Computer*
 MDT *Model Development Tools*
 MILS *Multiple Independent Levels of Security/Safety*
 MIT *Massachusetts Institute of Technology*
 MiXiM ... *MiXed siMulator*
 MOSA ... *Mission Oriented Sensor Array*
 MUAV ... *Mini UAV*
 NAV *Nano Air Vehicles*

NED *Network Description*
 NIST *National Institute of Standards and Technologies*
 OFP *Operational Flight Program*
 OMG *Object Management Group*
 OMU *Optical Measurement Unit*
 OSI *Open System Interconnection*
 PCB *Printed Circuit Board*
 PMD *Peso Máximo de Decolagem*
 PWM *Pulse Width Modulation*
 RFCSA ... *Reconfigurable Flight Control System Architecture*
 RPA *Remotely-Piloted Aircraft*
 RPAS *Remotely-Piloted Aircraft System*
 RPH *Remotely Piloted Helicopter*
 RPM *Rotações Por Minuto*
 RTOS *Real Time Operating Systems*
 SANT *Sistema de Aeronaves Não Tripuladas*
 SLUGS ... *Santa Cruz Low-cost UAV GNC System*
 SOA *Service Oriented Architectures*
 SPI *Serial Peripheral Interface*
 SSI *Smart Sensor Interface*
 StArt *State of the Art through Systematic Review*
 SWaP *Size, Weight, and Power*
 SysML ... *System Modelling Language*
 TCP *Transmission Control Protocol*
 TI *Tecnologia da Informação*
 TTCAN .. *Time-triggered Controller Area Network*
 TUAV *Tactical UAV*
 UAS *Unmanned Aerial Systems*
 UAV *Unmanned Aerial Vehicle*
 UCAP *Unmanned Air Vehicle Common Architecture Prototype*
 UCAR ... *Unmanned Combat Armed Rotorcraft*
 UCAV *Unmanned Combat Air Vehicle*
 UDP *User Datagram Protocol*
 UFSCAR . *Universidade Federal de São Carlos*
 UHF *Ultra High Frequency*
 UML *Unified Modeling Language*
 USArmy .. *United States Army*

VANETs .. *Vehicular Ad Hoc Networks*
VANT Veículo Aéreo Não Tripulado
VEINS ... *Vehicular in Network Simulation*
VHF *Very High Frequency*
VSP *Video Streaming Program*
VTOL *Vertical Take-Off and Landing*
VTUAV .. *Vertical Take-off UAV*
www *World Wide Web*
XML *eXtensible Markup Language*

SUMÁRIO

1	INTRODUÇÃO	33
1.1	Contextualização	33
1.2	Motivação e caracterização do problema	35
1.3	Objetivos	39
1.4	Contribuições	39
1.5	Organização da Tese	40
2	VEÍCULOS AÉREOS NÃO TRIPULADOS	41
2.1	Considerações iniciais	41
2.2	A Aeronave (ou Veículo Aéreo)	42
2.3	Estação de Controle Terrestre	47
2.4	O <i>Payload</i>	48
2.5	<i>Links</i> de Comunicação de Dados	48
2.6	Sistemas de Suporte	49
2.7	Processadores de Missão	50
2.8	Sistema de Navegação	50
2.9	Sistema de Controle	51
2.10	Considerações Finais	53
3	MODELOS ARQUITETURAIS	55
3.1	Considerações iniciais	55
3.2	Definição	55
3.3	Estado da arte	56
3.3.1	<i>Arquiteturas e modelos para veículos aéreos</i>	56
3.3.2	<i>A arquitetura IMA (Integrated Modular Avionics)</i>	68
3.4	Considerações finais	73
4	LARISSA	75
4.1	Considerações iniciais	75
4.2	O Modelo	75
4.3	Camada de Abstração do Sistema	78
4.4	Segmento Aéreo	79
4.4.1	<i>Camada Física do segmento Aéreo</i>	80
4.4.2	<i>Camada RTOS Distribuído</i>	85

4.4.3	<i>Camada de Monitoramento e Controle</i>	86
4.4.4	<i>Camada de Navegação e Serviços</i>	87
4.4.5	<i>Camada de Missão</i>	88
4.5	Segmento Terrestre	89
4.5.1	<i>Camada Física do segmento Terrestre</i>	90
4.5.2	<i>Camada Estação de Controle Terrestre</i>	90
4.6	Especificação do LARISSA	91
4.6.1	<i>UML</i>	91
4.6.2	<i>Metodologia utilizada para desenvolvimento do profile LARISSA</i>	92
4.6.3	<i>A modelagem do profile LARISSA</i>	98
4.7	<i>Template larissamodel2simulation</i>	103
4.7.1	<i>Template larissamodel2simulation para redes de computadores</i>	107
4.8	Considerações Finais	115
5	RESULTADOS EXPERIMENTAIS	117
5.1	Considerações Iniciais	117
5.2	Projeto dos experimentos da subcamada Comunicação	118
5.2.1	<i>Cenário I</i>	123
5.2.2	<i>Cenário II</i>	127
5.2.3	<i>Cenário III</i>	129
5.2.4	<i>Cenário IV</i>	133
5.2.4.1	<i>Experimento Real</i>	133
5.2.4.2	<i>Simulação</i>	135
5.3	Projeto dos experimentos da camada Controle	141
5.3.1	<i>SLUGS - Santa Cruz Low-cost UAV GNC System</i>	141
5.3.2	<i>YAPA - PAPARAZZI UAV</i>	149
5.4	Projeto dos experimentos da Camada Física do Segmento Aéreo	156
5.5	<i>AVENS - Aerial VEHICLE Network Simulator</i>	163
5.5.1	<i>Modelagem utilizando o profile LARISSA</i>	166
5.5.2	<i>Estudo de Caso</i>	170
5.6	Considerações Finais	174
6	CONCLUSÃO	177
6.1	Principais Resultados e Contribuições	178
6.2	Limitações do Modelo de Arquitetura LARISSA	179
6.3	Dificuldades Encontradas	179
6.4	Trabalhos Futuros	180
6.5	Declaração de autoria	181
6.6	Lista de publicações	182
6.6.1	<i>Artigos publicados</i>	182

6.6.2	<i>Artigos aceitos para publicação</i>	183
6.6.3	<i>Artigos submetidos</i>	183
REFERÊNCIAS		185
APÊNDICE A DOCUMENTO DE REQUISITOS		195
A.1	Introdução	195
A.1.1	<i>Objetivo do documento</i>	195
A.1.2	<i>Escopo do Produto: Projeto e desenvolvimento de um VANT capaz de realizar diversas aplicações dependendo do contexto.</i>	195
A.1.3	<i>Características do Usuário</i>	196
A.2	Descrição Geral do Produto	196
A.2.1	<i>Perspectiva do Produto</i>	196
A.2.2	<i>Requisitos Funcionais</i>	196
A.2.3	<i>Requisitos Não Funcionais</i>	197
A.2.4	<i>Requisitos de Desempenho</i>	197
A.2.5	<i>Requisitos de Segurança</i>	197
A.2.6	<i>Não requisitos</i>	198
APÊNDICE B REVISÃO SISTEMÁTICA		199
B.1	Planejamento	199
B.1.1	<i>Objetivos</i>	199
B.1.2	<i>Questões</i>	200
B.1.3	<i>Seleção das fontes</i>	201
B.1.4	<i>Seleção dos estudos</i>	201
B.2	Execução	203
B.2.1	<i>Construção da string de busca</i>	203
B.3	Seleção de estudos	203
APÊNDICE C ESPECIFICAÇÃO DO <i>PROFILE LARISSA</i>		207
APÊNDICE D CÓDIGO FONTE <i>LARISSAMODEL2SIMULATION</i>		217

INTRODUÇÃO

1.1 Contextualização

Os avanços tecnológicos e científicos vêm proporcionando diversas mudanças nos mais variados setores da indústria, comércio e serviços. Sistemas embarcados são sistemas computacionais que, de modo geral, fazem parte de um sistema maior, e sendo assim, proveem, em sua maioria, monitoramento e controle em tempo real para o sistema do qual fazem parte. A apresentação de requisitos especiais e o fornecimento de um conjunto predefinido de tarefas dedicadas a uma aplicação de tempo real são características dos sistemas embarcados (LAZIC; VELASEVIC, 2014). Esses sistemas são considerados críticos quando eventos de falha possibilitam perdas de vidas humanas ou de ativos de alto valor (DUNN, 2003; ARMOUSH; BECKSCHULZE; KOWALEWSKI, 2009; KUMAR; RAMAIAH; KHANAA, 2011; YI; CAI; YUE, 2008).

Tanto em hardware quanto em software os sistemas embarcados têm se tornado cada vez mais complexos¹. Sistemas *multicore* e multiprocessados são hoje comuns, o que tem aumentado ainda mais a complexidade do software (KOULOHERIS, 2003; WHITE, 2012). Em se tornando comuns, podem ser vistos tanto em ambientes domésticos como em ambientes profissionais, em que têm sido utilizados para o controle ou gerenciamento de informação. Graças aos avanços da tecnologia, os sistemas embarcados contam hoje com maior capacidade de processamento, memória e adaptação a diferentes necessidades, podendo inclusive se comunicar com qualquer outro dispositivo, seja ele embarcado ou não.

Um Veículo Aéreo Não Tripulado (VANT) constitui uma aplicação típica de um sistema embarcado crítico. O termo VANT foi adotado pela *Federal Aviation Administration* (FAA) e pela comunidade acadêmica internacional para designar a aeronave e o termo Sistema de Aeronaves Não Tripuladas (SANT) para designar o sistemas que incluem não apenas as aeronaves, mas

¹ Um sistema complexo é um sistema composto de vários componentes que podem interagir uns com os outros. Ele pode ser representado como uma rede onde os nós representam os componentes e as ligações suas interações.

todos os elementos associados, tais como o *payload*², a estação de controle terrestre e os *links* de comunicação (AUSTIN, 2010; DOD, 2008). VANTs têm sido amplamente utilizados em diversas áreas, como: agricultura de precisão, segurança nacional (missões militares) e monitoramento ambiental, entre outras. Diversos trabalhos já foram publicados demonstrando a viabilidade do uso desses veículos como ferramentas importantes para realização de diversas tarefas, como a agricultura de precisão e o monitoramento ambiental (AUSTIN, 2010; TRINDADE *et al.*, 2010; BRANCO *et al.*, 2011; TRINDADE *et al.*, 2012).

Existem diferentes tipos de VANTs que apresentam, inclusive, diferentes capacidades. Algumas aeronaves podem voar de forma autônoma, seguindo uma trajetória de voo pré-programada (baseada em um *grid* ou uma sequência de *waypoints*) (AUSTIN, 2010; TRINDADE *et al.*, 2010), enquanto outras voam recebendo comandos a partir de estações terrestres operadas por pilotos.

O tamanho da aeronave pode variar desde o micro até o macro, e a estação de controle terrestre pode ser implementada em *smartphones*, *tablets*, *notebooks* ou redes de estações de trabalho (estações de controle distribuídas) (AUSTIN, 2010). Desse modo, a aeronave pode variar não apenas em tamanho, mas também na forma, no tipo de propulsão e no desempenho.

A interface homem-máquina pode variar desde um *joystick* até uma interface de usuário tangível (por exemplo, uma mesa tangível com realidade aumentada). O desempenho dos *links* de comunicação e o tipo de carga também são muito importantes para que se possa cumprir a missão destinada ao sistema.

Literaturas especializadas citam que VANTs devem popular nos próximos 10 anos uma boa parte do espaço aéreo para a realização de diferentes missões, desde agrícolas passando por inspeções de fronteira chegando até ao transporte automático de cargas, sejam essas cargas leves ou pesadas (desde uma simples pizza a uma carga de uma empresa) (BRANCO, 2012; RODRIGUES *et al.*, 2011; AUSTIN, 2010; DOD, 2009; DOD, 2007; DOD, 2005; DOD, 2002). Sendo assim, acredita-se que em um futuro um pouco mais distante, que as únicas aeronaves tripuladas que deverão permanecer atuantes são as que transportam passageiros, a exemplo do que ocorre em outros sistemas de transporte, por exemplo, o metroviário onde o condutor precisa atuar somente em emergências.

Esse uso crescente dos VANTs deve fazer com que eles se tornem comuns, passando a ser comercializados de forma mais ampla. Nesse cenário, arquiteturas que permitam a organização e a definição mais específica dos componentes que compõem esses sistemas (os SANTs) facilitam o desenvolvimento de hardware e software que os compõem, permitindo que esses veículos possam ser inseridos e incorporados mais facilmente ao espaço aéreo e contribuindo para a sua disseminação.

² Sensores utilizados para obter informações orientados pelas necessidades da missão, como câmeras de vídeo e fotográficas; sensores infravermelhos e térmicos; e radares.

Os componentes de um SANT podem ser divididos em segmentos aéreo e terrestre e cada um desses segmentos pode ser subdividido. A divisão desses segmentos em camadas permite que o sistema seja visto como subsistemas que possam ter implementações diferentes, auxiliando a separação das partes que compõem um sistema complexo em diferentes níveis de criticidade. Estas camadas podem ser representadas por meio de modelos, os quais tem o propósito de servirem de guias para elaboração dos SANTs, especificando como se dá a interligação entre os diversos componentes tais como sensores, circuitos de controle, *Global Positioning System* (GPS) - sistema de posicionamento global, *payload*, comunicação com a estação de controle terrestre, dentre outros.

Em Tecnologia da Informação (TI), uma arquitetura em camadas é utilizada para definir o funcionamento específico de cada camada e a interligação entre elas. As mesmas vantagens podem ser obtidas no cenário de desenvolvimento de SANTs. Baseado em um modelo de arquitetura, o fabricante de hardware ou o projetista de software pode desenvolver seus produtos sabendo exatamente em que camada ele irá interagir no SANT, quais são os parâmetros de entrada e saída ou até mesmo qual o tipo de conexão deve ser utilizada.

Nesta tese é apresentado o *Layered ARchitecture model for Interconnection of Systems in uAs* (LARISSA). O LARISSA é um modelo de arquitetura em camadas com alto nível de abstração, que visa a padronizar os componentes do domínio SANT, com o objetivo de permitir que pesquisadores, desenvolvedores e fabricantes produzam componentes individuais, os quais possam ser acoplados aos demais, independente de quem os fabricou e gerar subsídios para certificação perante os órgãos governamentais competentes, seja do VANT, do SANT, ou de seus componentes.

1.2 Motivação e caracterização do problema

Um fato observado é que os VANTs estão cada vez mais comuns e sua disseminação civil teve um crescimento nos últimos anos ([ANAC, 2015a](#)) em muito impulsionado pelos quadricópteros utilizados seja para fins de recreação ou profissional. Esse crescimento tem levado as agências reguladoras, tais como a FAA nos Estados Unidos da América (EUA), Agência Nacional de Aviação Civil (ANAC) no Brasil e muitas outras de vários países, a regulamentar o uso destas aeronaves no espaço aéreo segregado e não segregado. A ANAC propôs no final de 2015 uma regulamentação para o setor ([ANAC, 2015b](#)), a qual foi à consulta pública, mas ainda não foi regulamentada. Dentre os itens desta proposta de regulamentação consta a classificação das aeronaves tomando como parâmetro o peso de decolagem. Para as aeronaves com peso menor que 25 kg preconiza-se a não exigência de certificação, enquanto que para as de 25 a 150 kg faz-se necessária simplesmente a apresentação do projeto da aeronave e a certificação na etapa da produção. Já as aeronaves maiores que 150 kg, assim como as aeronaves tripuladas, deverão ser submetidas ao processo de certificação completo.

Entretanto, é de conhecimento dos profissionais aeronautas, que a certificação de uma aeronave não é trivial e todos os componentes utilizados além do próprio processo de fabricação desta devem passar pelo processo de certificação (ANAC, 2010; ANAC, 2013a; ANAC, 2013b). Segundo a ANAC (2010) o processo de certificação de uma aeronave passa por diversas fases, destacando-se entre elas a Certificação de Projeto, Certificação de Motores e Hélices, Certificação de Produção e Vigilância, e Certificação da Aeronavegabilidade. Na Fase de certificação do Projeto o uso de simulações em computadores é muito bem aceito como uma das fontes de dados utilizadas para compor o documento de avaliação da engenharia. Além disso, outro fato diz respeito a padronização dos componentes que compõem um VANT: como ocorre em novas tecnologias, cada fabricante utiliza seus métodos e especificações sem se preocupar com padrões que permitam utilizar componentes fabricados por terceiros. Isso dificulta ainda mais o processo de certificação e consequentemente o de regulamentação dessas aeronaves.

Elementos tais como (i) a quantidade crescente de VANTs em espaço aéreo segregado e não-segregado, (ii) o aumento do número de diferentes e simultâneas missões a serem realizadas por um ou mais VANTs, (iii) os problemas de *security* e de *safety*³ relacionados com a inserção dos SANTS no espaço aéreo, frisando que para sistemas que implementem funções de *safety* de diferentes níveis de integridade, e onde não se possa demonstrar que há independência de implementação entre essas funções, as partes do hardware e software onde a independência de execução é mínima, deve ser tratada como pertencente à função de *safety* com o mais alto nível de integridade. Por conseguinte, os requisitos aplicáveis ao nível mais elevado de integridade de *safety* devem aplicar-se a todas as partes (IEC, 2010). E (iv) a necessidade de certificação inerente a integração desses veículos, e que devem passar a ser uma exigência a partir do amplo crescimento e da grande utilização de VANTs, caracterizam as dificuldades do desenvolvimento de SANTS com vistas a facilitar sua inserção no espaço aéreo. Pesquisas existentes nesta área estão a procura de soluções para: integrar os SANTS no espaço aéreo segregado e não-segregado; garantir *security* e *safety*; desenvolver modelos que permitam reutilizar componentes de software de SANTS; desenvolver ou melhorar modelos de arquitetura existentes para SANTS; e usar abordagens não-deterministas. No caso específico deste trabalho há o empenho em compreender as vantagens do uso de modelos de arquiteturas e de seu comportamento quando aplicado a cenários de SANTS.

Existem *roadmaps* publicados periodicamente por organizações militares e civis (por exemplo, *United States Army* (USArmy), *Department of Defense* (DoD), *European RPAS Steering Group* (ERSG), ou da *Federal Aviation Administration* (FAA)) que ilustram os avanços esperados para os SANTS (ARMY, 2010; AUTHORITY, 2012; DOD, 2013; YEARBOOK, 2011). Com base na análise detalhada destes *roadmaps*, foi possível identificar uma lista de desafios relacionados com a integração de SANTS no espaço aéreo não-segregado, nas áreas de regulamentação, investigação/pesquisa e impacto social. Esses desafios, ilustrados na Figura 1,

³ Neste contexto, *safety* refere-se a saúde dos componentes físicos

são classificados em oportunidades e necessidades e parte desses desafios podem ser enfrentados por meio da abordagem apresentada nesta tese.

Figura 1 – Desafios de integração dos SANTS no espaço aéreo: Oportunidades e Necessidades.

Fonte: Elaborada pelo autor.

Esses desafios podem ser abordados por meio de uma estrutura que pretende mitigar as necessidades de integração de SANTS no espaço aéreo e, ao mesmo tempo, faz uso das oportunidades existentes. Um modelo de arquitetura em camadas vem ao encontro da mitigação de *security* e *safety*, regulamentação de segurança e certificação, uma vez que facilita a integração dos SANTS no espaço aéreo, bem como a reutilização (proporcionando modularidade e facilidade de manutenção) das várias partes que compõem um SANT.

Um modelo de arquitetura em camadas auxilia em duas frentes: na definição do modelo em si e sua separação em camadas distintas; e na construção de SANTS a partir de componentes previamente certificados.

A adoção do modelo guia a fabricação, manutenção, utilização e reutilização de componentes de um SANT, facilitado pela separação em camadas, a qual permite que o sistema seja dividido em subsistemas que podem ter implementações diferentes auxiliando na separação das partes que compõem um sistema embarcado crítico complexo. Essa separação pode ser feita com base nos diferentes níveis de criticidade, que por sua vez auxiliam no desenvolvimento e manutenção individual dos componentes de cada camada e permitem que um componente previamente desenvolvido e testado possa ser utilizado em um novo projeto independente de fabricante, ou ainda permitem que um novo fabricante venha produzir um determinado componente e este se conecte a componentes fabricados por terceiros.

Quando da construção de um novo SANT, sendo os componentes individuais previamente certificados, diminui-se o tempo para certificação do mesmo, o que permite que este entre em operação em um menor espaço de tempo.

Com base na crescente utilização dos VANTs em aplicações civis, aliada aos desafios e lacunas apresentadas nos *roadmaps* que demonstram uma clara necessidade de integração dos SANTs ao espaço aéreo, é possível vislumbrar:

Problema: Existe uma maneira de facilitar ou viabilizar a inserção de SANTs no espaço aéreo em sua etapa de desenvolvimento?

Hipótese: Com o uso de um modelo de arquitetura em camadas será possível aumentar a modularização do sistema, facilitando a identificação de possíveis problemas de comunicação por isolamento e facilitando o reuso e manutenção de componentes.

Partindo da realidade que se vivencia nos dias de hoje, o objetivo deste trabalho é especificar, desenvolver e validar um modelo de arquitetura em camadas, considerando-se que o uso desse modelo possa vir a facilitar o desenvolvimento de SANTs e sua integração no espaço aéreo.

Quando da existência de um modelo pode-se, a partir deste, desenvolver um hardware ou software que implemente as interfaces e possa assim, ser conectado aos demais componentes já existentes no mercado obedecendo as especificidades deste modelo (Entretanto, devido serem elementos que necessitam de um melhor desenvolvimento, seus detalhes não são tratados neste trabalho). Em outras palavras, isto significa que um novo tipo de câmera pode ser acoplado a um SANT já existente, desde que essa tenha sido projetada obedecendo o modelo ou que possua uma interface compatível com o mesmo. O mesmo é válido para os softwares, sejam eles utilizados no interior dos VANTs ou nas estações de controle terrestre. Cada um desses componentes pode ser testado e analisado independentemente de modo a serem certificados individualmente.

O conceito de desenvolvimento de SANTs obedecendo o modelo de arquitetura de camadas também traz benefícios no desenvolvimento de projetos que envolvem vários pesquisadores e instituições, uma vez que de posse do modelo, cada um pode desenvolver seu circuito, seu sensor, seu hardware ou seu software com a garantia que ele se conectará aos componentes desenvolvidos pelos parceiros.

Tudo isso parece um tanto simples, mas é, na verdade, muito complexo quando considerada a especificação, implementação e validação propriamente ditas. O aspecto mais difícil a ser contemplado é a integração das diferentes áreas, de modo a permitir uma validação que realmente permita que o modelo possa ser considerado válido e eficiente.

Neste sentido o objetivo desta tese é a definição e validação de um modelo de arquitetura em camadas para interconexão de sistemas em SANT.

1.3 Objetivos

O objetivo principal deste trabalho é prover um modelo de arquitetura para Sistemas Aéreos Não Tripulados, baseado em camadas de modo a definir as interfaces e a comunicação entre essas camadas estabelecendo um padrão para interconexão de sistemas utilizados em um SANT.

O quesito *safety* é em parte contemplado pelo modelo, pois este deve garantir que as funções distintas de cada nível sejam ali executadas e a comunicação entre os níveis deve ocorrer com o uso de interfaces, comunicando somente com a camada superior e inferior adjacente. Esse quesito auxilia na obtenção de certificação dos SANTs para sua inserção no espaço aéreo.

Objetivos secundários podem ser identificados a partir do objetivo maior desta tese:

1. Definir o modelo em uma linguagem que facilite seu uso pelos desenvolvedores;
2. Utilizar e se possível desenvolver simuladores, que façam uso do modelo e gerem subsídios para o processo de certificação de SANT;
3. Elaborar cenários de testes com a finalidade de validar o modelo;
4. Desenvolver ferramentas para auxiliar na validação dos cenários.

1.4 Contribuições

Nesta tese, os desafios apresentados em seção anterior são tratados de modo a ter as seguintes contribuições:

- **profile LARISSA:** o *profile* LARISSA é a materialização do modelo de arquitetura em uma linguagem padrão, a SysML, que permite a construção dos projetos de SANT propriamente ditos (MARCONATO *et al.*, 2016).
- **Simulador AVENS:** um simulador de redes de VANTs com a geração automática de código produzido a partir do desenvolvimento do modelo de arquitetura LARISSA (MARCONATO *et al.*, 2017).
- **Uso de simulação para certificação de SANTs:** demonstrou-se, assim como apregoa a ANAC (ANAC, 2010) o uso de simulação para validar o modelo de arquitetura e consequentemente prover dados para fases de certificação (SILVA; MARCONATO; BRANCO, 2015).
- **larissamodel2simulation:** desenvolvimento do *larissamodel2simulation* que permite a geração automática de código para o OMNeT++ (OMNET, 2014) garantindo a fidelidade

ao modelo e evitando falhas na transcrição entre modelo e simulador ([MARCONATO et al., 2016](#)).

- **LARISSA**: especificação e desenvolvimento de um modelo de arquitetura em camadas. Esse modelo permite que pesquisadores, desenvolvedores e fabricantes façam uso da arquitetura em seus projetos, utilizando a UML. A divisão do modelo de arquitetura em camadas traz benefícios no que se refere a restrição das funções a cada camada, levando à padronização, facilitando a manutenção e o reúso de artefatos que compõem o SANT ([MARCONATO et al., 2014](#); [MARCONATO et al., 2014](#); [MARCONATO; BRANCO, 2014](#)). O modelo em camadas facilita o uso de Arquiteturas Orientadas a Serviço em SANTs ([RODRIGUES et al., 2016](#)) e a possibilidade de se identificar problemas devido ao isolamento proporcionado por uma arquitetura em camadas.

1.5 Organização da Tese

- No [Capítulo 2](#) são descritos os principais conceitos relacionados a VANTs e SANTs, elucidando suas principais aplicações;
- O [Capítulo 3](#) apresenta uma revisão bibliográfica sobre os Modelos Arquiteturais utilizados nos SANTs, vislumbrando o estado da arte;
- A proposta do modelo de arquitetura de camadas é abordada e discutida no [Capítulo 4](#);
- O [Capítulo 5](#) descreve os cenários, testes e resultados conduzidos para validar o modelo proposto;
- Por fim, o [Capítulo 6](#) traz a conclusão da tese, assim como as produções bibliográficas e proposta para futuros trabalhos.

VEÍCULOS AÉREOS NÃO TRIPULADOS

2.1 Considerações iniciais

Dentro da categoria de robôs móveis, uma que vem se destacando no cenário nacional atualmente é a dos robôs aéreos. Entende-se por robô um dispositivo, ou grupo de dispositivos eletromecânicos ou biomecânicos capazes de realizar trabalhos de maneira autônoma, pré-programada, ou por meio de controle humano, ou seja, um agente artificial ativo cujo ambiente é o mundo físico (DUDEK; JENKIN, 2000; MATARIC, 2007).

Definido em 1990, o termo robô aéreo descreve uma classe de máquinas voadoras pequenas e inteligentes, não impondo nenhuma restrição de autonomia, podendo ser autônomas ou controladas a distância por operadores humanos. As aeronaves mais utilizadas são os aviões, helicópteros e dirigíveis, existindo ainda propostas de "aeronaves não convencionais" (NONAMI *et al.*, 2010).

Um VANT - do inglês *Unmanned Aerial Vehicle* (UAV) - é um robô aéreo que faz parte de um SANT. Esse termo é utilizado pelo Departamento de Defesa para refletir que esses sistemas complexos incluem não somente a aeronave, mas também as estações de solo, o *payload* e outros elementos (NONAMI *et al.*, 2010).

Os VANTs podem basicamente ser considerados veículos aéreos que não possuem um ser humano pilotando a aeronave de forma convencional. O VANT pode ser pilotado a distância por meio de uma estação de controle terrestre ou realizar missões pré-programadas de forma autônoma (TRINDADE *et al.*, 2002),

Segundo Trindade *et al.* (2010) e Austin (2010), os VANTs podem ser aeronaves das mais diversas formas, como já mencionado anteriormente, tais como: asas fixas (aviões), asas rotativas (helicópteros), quadricópteros, balões dirigíveis e variantes de todos estes, além de utilizar os mais diversos meios de propulsão: combustão interna, turbinas, motores elétricos, gases mais leves que o ar, entre outros. Além disso, o emprego dessas aeronaves para fins

militares pode variar desde um alvo móvel para treinamento de artilharia antiaérea, vigilância, reconhecimento, monitoramento até as versões mais avançadas como aeronaves armadas para ataque. As versões civis são utilizadas para fotografia aérea, filmagem, busca e salvamento, monitoramento ambiental e recreação entre várias outras que surgem a cada dia.

O VANT faz parte de um SANT - do Inglês *Unmanned Aerial Systems* (UAS) que pode ser dividido em (AUSTIN, 2010):

- A **aeronave**, dotada de processadores de missão, sistemas de navegação e sensores.
- **Estação de Controle Terrestre**, conhecida também como estação base, constitui o centro de controle operacional de todo o sistema. É a partir dela que se pode controlar desde o lançamento, passando pelo voo ou operação propriamente dita até chegar à recuperação do VANT. Fica a cargo dela também receber e processar os dados dos sensores que compõem o *payload*, controlar o seu funcionamento (geralmente em tempo real) e fornecer uma interface entre o sistema e o mundo exterior. Ela é um sistema de computador (que pode variar desde um *smartphone* até um super computador) projetado para acompanhar o desenvolvimento da missão e, eventualmente, operar o VANT e seu *payload* (PASTOR; LOPEZ; ROYO, 2007).
- **Payload**, normalmente sensores, e equipamentos adicionados com a finalidade de realizar uma missão. O *payload* pode também englobar armamentos e outros dispositivos, excluindo-se aviônicos (eletrônicos da aviação), *links* de comunicação e combustível (PASTOR; LOPEZ; ROYO, 2007; TRINDADE *et al.*, 2012).
- **Links de Comunicação de Dados**, uma combinação de mecanismos de comunicação (rádio, satélites, dentre outros) que deve garantir conexão contínua entre o VANT e a estação de controle terrestre (PASTOR; LOPEZ; ROYO, 2006). Compõem a parte do *link* o hardware necessário para a comunicação, as antenas de comunicação, normalmente com opção de rastreamento da aeronave para aumento de alcance e largura da banda de comunicação;
- **Sistemas de suporte**, o que incluem sistemas de manutenção, lançamento, pouso e transporte da aeronave (AUSTIN, 2010).

2.2 A Aeronave (ou Veículo Aéreo)

Um VANT não deve ser confundido com um modelo rádio controlado, que é uma aeronave utilizada para prática de *hobby* ou esporte e onde seu operador restringe-se a emitir comandos para a aeronave subir, descer e virar para os lados, um VANT vai além disso. Os VANTs possuem um certo grau de inteligência e um conjunto de sensores que os permite transmitir à estação de controle terrestre informações tais como: altitude, velocidade, posição,

imagens e condições de funcionamento da aeronave (como nível de combustível, temperatura do motor, temperatura de componentes eletrônicos, nível de carga de baterias, Rotações Por Minuto (RPM) do motor, assim como receber instruções da estação de controle, tais como novas altitudes e velocidades ou novo plano de voo.

Já os *Drones*, apesar de muitas vezes serem utilizados como sinônimos para VANTs, possuem um processador que é pré-programado para executar uma determinada missão, mas não possui qualquer nível de comunicação com uma estação de controle terrestre e os resultados de fotos, por exemplo, são conhecidos somente após a aeronave retornar ao solo (AUSTIN, 2010). Essa é uma diferença sutil, mas que impõe uma diferença significativa de capacidades e de tecnologias.

Dessa forma, a parte do VANT denominada de Veículo Aéreo ou Aeronave é composta principalmente pela estrutura, grupo motopropulsor, acionadores, sistema de guiamento e controle e subsistemas de comunicação (TRINDADE *et al.*, 2012; NONAMI *et al.*, 2010). Essas características da aeronave acabam por definir uma forma de classificação dos VANTs, dado que, as especificações da aeronave são fatores determinantes no seu propósito de utilização e seu raio de ação. A importância desse tipo de classificação nesse trabalho dar-se-á pelo fato de serem necessários o levantamento e a determinação de todas as especificidades dos VANTs para que se possa incorporar a maioria dos diferentes tipos de aeronaves (seja por classe, propósito de utilização) no modelo de arquitetura proposta.

Uma classificação possível para os VANTs é a proposta em DOD (2009). A classificação proposta é dada segundo os recursos, tamanhos e capacidade. Nesse sentido, essa classificação divide os VANTS em seis categorias funcionais e quatro relacionadas ao tamanho e capacidade da aeronave. As categorias funcionais são:

- **Alvo:** representando aeronaves que servem como alvo para outra aeronave ou um míssil;
- **Reconhecimento:** aeronaves destinadas a batalhas inteligentes;
- **Combate:** aeronaves com capacidade de ataque para missões de alto risco (combates aéreos de veículos não tripulados);
- **Logística:** aeronaves projetadas especificamente para o transporte de cargas e operações logísticas;
- **Pesquisa e Desenvolvimento:** aeronaves utilizadas para desenvolver tecnologias que serão utilizadas em outras categorias de VANTS;
- **Civil e Comercial:** aeronaves especificamente projetadas para aplicações civis e comerciais.

Com relação ao tamanho e capacidade, as quatro classes são (DOD, 2007):

- **Pequenas:** peso bruto de decolagem inferior a 55 Kg;
- **Táticas:** peso bruto de decolagem entre 55 e 1.320 Kg;
- **Theater:** peso bruto de decolagem superior a 1.320 Kg;
- **Combate:** projetadas desde o início como uma plataforma de ataque com bombas, armas e equipamentos para sobrevivência. Peso bruto de decolagem maior que 1.320 Kg.

Uma vez apresentadas algumas das formas a partir das quais os VANTs podem ser classificados, é possível observar que a classificação é uma das dificuldades relacionadas ao assunto VANT/regulamentação/certificação. Isso se deve, principalmente, à falta de padronização de uma metodologia na classificação desse tipo de aeronave.

Essa ausência de normas que possa ser adotada internacionalmente para a classificação permite que cada país desenvolva suas próprias regras para categorizar os VANTs. Desse modo, várias são as classificações utilizadas nos diversos países como Reino Unido, Estados Unidos, Brasil e Europa em geral.

No Reino Unido, a proposta sobre a regulamentação de VANTs encontra-se na sexta revisão. O assunto é tratado pelo *Safety and Airspace Regulation Group* - Grupo de Segurança e Regulamentação do Espaço Aéreo da *Civil Aviation Authority* (CAA) (CAA, 2015). A classificação utilizada leva em consideração o peso da aeronave, conforme [Quadro 1](#).

Quadro 1 – Classificação RPAS/RPA - proposta Reino Unido.

Categoria	Peso (kg)	Órgão Regulador
VANT Pequeno	0 - 20	<i>National Aviation Authority</i>
VANT Leve	> 20 até 150	<i>National Aviation Authority</i>
VANT	> 150	<i>European Aviation Safety Agency</i>

Fonte: Adaptada de CAA (2015).

A classificação dos Estados Unidos é feita pelo DOD (2009) e é baseada nos atributos de peso máximo de decolagem, altitude de voo e velocidade, fornecendo assim uma referência para o agrupamento dos VANTs, como descrito no [Quadro 4](#).

No Brasil, a Agência Nacional de Aviação Civil (ANAC) propôs um novo regulamento para os VANTs (ANAC, 2015b), entretanto, até o término da escrita desta tese este ainda não havia entrado em vigor. Algumas das principais definições propostas são:

- O termo adotado pela ANAC para denominar o VANT é Aeronave Remotamente Pilotada - *Remotely-Piloted Aircraft* (RPA);
- Para o SANT o termo é Sistema de Aeronave Remotamente Pilotada - *Remotely-Piloted Aircraft System* (RPAS) é o conjunto de todos os elementos o que inclui a aeronave, a

estação de controle terrestre, o enlace de dados e os demais itens necessários à operação do RPA;

Com relação à classificação do RPAS e RPA, a ANAC sugeriu que sejam classificados de acordo com o Peso Máximo de Decolagem (PMD), sendo previamente definidos como consta no [Quadro 2](#).

Quadro 2 – Classificação RPAS/RPA - proposta ANAC

Classificação	PMD (kg)
Classe 1	> 150
Classe 2	> 25 até 150
Classe 3	<= 25

Fonte: Adaptada de [ANAC \(2015b\)](#).

Por fim, a classificação Europeia, proposta pela *European Aviation Safety Agency* (EASA) ([EASA, 2015](#)), foi realizada de acordo com o risco de voo proporcionado pelos VANTs, que pode ser baixo, médio ou alto. Cada uma dessas categorias é subdividida considerando-se como parâmetros: alcance, altitude de voo, autonomia de voo e peso máximo de decolagem. Um resumo dessa classificação é exibido no [Quadro 3](#).

Quadro 3 – Classificação RPAS/RPA - proposta Europeia

Classificação	Risco	Observação
<i>Open</i>	Baixo	Limitação de teto operacional e de peso
<i>Specific</i>	Médio	Necessita de autorização da <i>National Aviation Authority</i> . Necessita de manual de operações
<i>Certified</i>	Alto	Necessita de autorização da <i>National Aviation Authority</i> . Requisitos idênticos da aviação convencional

Fonte: Adaptada de [EASA \(2015\)](#).

Os VANTs são utilizados para realização de missões consideradas entediadas (*dull*), sujas (*dirty*) e perigosas (*dangerous*), conhecidas pela sigla em inglês *Dull, Dirty, Dangerous* (DDD) ([AUSTIN, 2010](#); [PASTOR; LOPEZ; ROYO, 2007](#)).

Missões entediadas, por exemplo, referem-se à vigilância, normalmente de longa duração que podem levar a tripulação da aeronave à fadiga e à perda de concentração e, conseqüentemente, à menor eficácia da missão. Um VANT não é afetado por estes efeitos, uma vez que o piloto é automático, ou seja, é uma máquina que não está sujeita a esse tipo de fadiga (como é o caso do ser humano).

Missões sujas são aquelas em que existe a possibilidade de contaminação, como por exemplo, o sobrevoo de uma usina nuclear com vazamento radioativo (contaminação radio-

ativa) ou áreas afetadas por contaminação biológica. Uma aeronave não tripulada pode ser descontaminada após a realização da missão.

As missões perigosas são, por exemplo, os casos de missões militares, onde a tripulação é constantemente submetida a situações perigosas, colocando em risco vidas humanas. Caso estas aeronaves sejam substituídas por VANTs o risco de perda da tripulação é praticamente eliminado. Esse parâmetro, a missão, também pode ser considerado para a classificação dos VANTs, como é o caso da classificação apresentada por [Austin \(2010\)](#), ilustradas no [Quadro 4](#).

Quadro 4 – Classificação conforme propósito de utilização e raio de atuação.

Tipo	Características	Alcance	Propósito
<i>High altitude long endurance</i> (HALE)	Altitude: maior que 15.000 m Tempo de voo: mais que 24 h <i>Payload</i> : 860 Kg	Global	Militar: Reconhecimento e Vigilância
<i>Medium altitude long endurance</i> (MALE)	Altitude: 5.000 - 15.000 m Tempo de voo: 24h <i>Payload</i> : 200 Kg	Até 500 Km	Militar: Reconhecimento e Vigilância
<i>Tactical UAV</i> (TUAV) or <i>Medium Range</i>	São aeronaves pequenas, operadas por sistemas mais simples que os dos HALEs e MALEs Tempo de voo: 5 h <i>Payload</i> : 25 Kg	100 - 300 Km	Militar: Reconhecimento e Vigilância
<i>Close-Range UAV</i>	Idênticas os TUAVs	Até 100 Km	Militar: Grupos de Combate, reconhecimento e designação de alvos Civil: Propósitos Gerais
Mini UAV (MUAV)	Pesar até 20 Kg e ser maior que um MAV	Até 30 Km	Militar: Grupos de Combate Civil: Propósitos Gerais
Micro UAV (MAV)	Não ter as asas com mais de 1,5m e ser lançado manualmente	Urbano	Militar e Civil: Propósitos Gerais
<i>Nano Air Vehicles</i> (NAV)	Tamanho reduzido, capaz de voar entre construções ou até em seu interior	Espaços restritos	Militar: Interferência eletrônica, Vigilância Civil: Vigilância

Fonte: Adaptada de [Austin \(2010\)](#).

Alguns VANTs compreendidos entre o HALEs e TUAVs podem ser classificados como *Remotely Piloted Helicopter* (RPH) ou *Vertical Take-off UAV* (VTUAV) e as versões de combate

(armadas) são designadas como *Unmanned Combat Air Vehicle* (UCAV) e *Unmanned Combat Armed Rotorcraft* (UCAR). Os modelos típicos de VANTs são ilustrados na [Figura 2](#), o que permite compreender e associar os conceitos mencionados no [Quadro 4](#).

Figura 2 – Exemplos de Tipos de VANTs

Fonte: Adaptada de [Nonami et al. \(2010\)](#).

2.3 Estação de Controle Terrestre

A estação de controle terrestre ou *Ground Control Station* (GCS) é um sistema de computação em solo que monitora a execução da missão, recebendo informações do *payload* (imagens, fotos), dados de telemetria da aeronave - tais como temperatura do motor, nível de combustível, altitude, velocidade - e gerencia o sistema de comunicação. Há estações de controle terrestre que possuem a capacidade de operar (pilotar) remotamente o VANT, assim como seu *payload* (girar as câmeras, acionar um sistema pulverização, sistema de armamento) ([ARMY, 2010](#); [NONAMI et al., 2010](#)).

A estação de controle terrestre pode variar desde um *smartphone*, passando por um *notebook* até um complexo conjunto de veículos com sistemas de computadores, monitores, controles de voo e de *payload*, sistemas de comunicação (*links* de rádio, antenas). É na estação de controle terrestre que ficam os operadores do VANT que pode variar de uma a duas pessoas (uma para controlar o VANT e outra para operar o *payload* - quando necessário) (AUSTIN, 2010; PASTOR; LOPEZ; ROYO, 2007; ARMY, 2010).

2.4 O Payload

O *payload* é definido por Pastor, Lopez e Royo (2007) como sendo o conjunto de sensores, composto de câmeras de TV, câmeras fotográficas, sensores infravermelhos, sensores térmicos, radares e outros com a finalidade de obter informações que podem ser parcialmente processadas internamente ou transmitidas para estação de controle terrestre para análise. Entretanto, segundo Austin (2010), o *payload* é orientado pelas necessidades das tarefas ou missões que o VANT deve realizar. Em VANTs de aplicações civis, por exemplo, o *payload* pode ser, além dos dispositivos já citados, um sistema de pulverização para fins agrícolas ou um sensor para medir nível de poluição.

A execução das missões realizadas pelos VANTs possui uma dependência elevada com seu *payload* que em sua maioria são modulares, o que possibilita sua substituição de acordo com a necessidade da missão. Não são raros os casos em que os VANTs são projetados levando-se em consideração as necessidades do *payload* que precisa ser embarcado para o cumprimento de uma missão específica (AUSTIN, 2010).

Os dados coletados pelos sensores do *payload* são utilizados para detectar, reconhecer e identificar, medir e classificar entidades e eventos e obter informações úteis sobre o ambiente ao seu redor (NIST, 2002; AUSTIN, 2010). Sendo os sensores somente responsáveis pela coleta de dados, a tarefa de processar esses dados é realizada nas estações de controle terrestre, nos processadores de missão, em processadores específicos ou então particionada de acordo com as características específicas de cada missão/aeronave.

2.5 Links de Comunicação de Dados

Um *link* ou enlace de comunicação de dados, é uma arquitetura de hardware e software que permite ao VANT transmitir e receber dados da e para a estação de controle terrestre. O sistema de comunicação pode ser uma mistura de diferentes mecanismos como rádio modems, micro-ondas e comunicação via satélites que garantam a comunicação contínua entre o VANT e estação de controle terrestre (PASTOR; LOPEZ; ROYO, 2007).

A manutenção de um *link* de dados entre o VANT e a estação de controle terrestre é primordial, pois sem esta o VANT passa a ser somente um *Drone* (AUSTIN, 2010). A perda de

comunicação durante a operação pode ser resultado de:

- falha total ou de parte do sistema, devido à falta de confiabilidade;
- perda de visada - *line-of-sight* (LOS), devido a características geográficas que bloqueiam os sinais de rádio;
- enfraquecimento do sinal recebido, devido à grande distância entre o VANT e a estação de controle terrestre;
- interferência intencional ou inadvertida de sinais de rádio.

Os *links* de comunicação carregam indiretamente um quesito importante, a segurança. Uma vez que toda a transmissão de informações feita entre aeronave e estação de solo se dá por meio da estação de solo, é imprescindível que haja uma garantia da confiabilidade desse meio de transmissão.

2.6 Sistemas de Suporte

Assim como uma aeronave convencional, um VANT necessita de sistemas de suporte que proveem toda infraestrutura necessária a seu funcionamento (AUSTIN, 2010; PASTOR; LOPEZ; ROYO, 2007; ARMY, 2010), dentre eles destacam-se:

- sistema de lançamento (catapulta) e recuperação (paraquedas, redes);
- sistema de transporte do VANT, estação de controle terrestre e demais equipamentos;
- sistema de resuprimento - reabastecimento de combustível, recarga de baterias, equipamento de gravação de dados, óleo e fluído de motor, peças sobressalentes;
- sistema de substituição de peças do VANT ou do *payload*;
- ferramentas para manutenção;
- montagem do sistema de comunicação (antenas).

Na [Figura 3](#) é ilustrado um VANT com sistema de suporte (antenas de comunicação, catapulta e veículos de transporte).

Além das partes em que [Austin \(2010\)](#) divide um SANT, faz-se necessário o detalhamento de alguns outros subsistemas dos VANTs, elencados nas seções seguintes.

Figura 3 – VANT e seu sistema de suporte

Fonte: [Austin \(2010\)](#)

2.7 Processadores de Missão

Basicamente é um computador de bordo cuja tarefa é orientar a execução da missão, seguindo o plano de voo, controlando o funcionamento do *payload* ([PASTOR; LOPEZ; ROYO, 2007](#); [NONAMI et al., 2010](#)). Esses processadores são também responsáveis por processar dados captados pelos sensores sendo assim capazes de detectar, reconhecer e identificar entidades, baseados em características e atributos que são comparados com modelos do mundo real armazenados previamente ([NIST, 2002](#)). O processador de missão está ligado ao sistema de navegação ou computador de voo. Isso se deve ao fato de ser ele o responsável por passar ao sistema de navegação informações importantes de posicionamento e realização da missão. Cabe ressaltar que o processador de missão, funciona como o componente que indica o funcionamento da aeronave, possui prioridade menor que o processador responsável pelo sistema de controle da aeronave. Desse modo, a qualquer instante os comandos efetuados pelo processador de missão podem ser suprimidos por um comando de mais alto nível de prioridade oriundo do sistema de controle.

2.8 Sistema de Navegação

O sistema de navegação é constituído por uma série de sensores projetados para coletar informações aerodinâmicas e de posicionamento tais como GPS, magnetômetros, acelerômetros, sensores de pressão, giroscópios e outros. Estas informações são enviadas a um processador ou computador de voo que aciona as superfícies de controle do VANT fazendo com que este siga o plano de voo ([PASTOR; LOPEZ; ROYO, 2007](#); [NONAMI et al., 2010](#)).

De modo geral, os sistemas de navegação mais comuns nos VANTS recebem como

entrada:

- um plano de voo, que consiste em um programa escrito em uma linguagem especificando a missão que se deseja que o VANT desempenhe. Trata-se de uma entrada que deve ser fornecida antes da decolagem;
- uma região autorizada, que consiste na especificação de um polígono georeferenciado ao qual todo plano de voo válido deve estar inscrito;
- dados de sensores, produzidos pela Unidade Barométrica e a Unidade Inercial, entre os quais se destacam (i) altitude, (ii) posição geográfica, (iii) velocidade aerodinâmica e (iv) velocidade em relação ao solo. Trata-se de uma série temporal, com a qual o sistema de navegação é alimentado continuamente durante o voo;
- sinais secundários diversos de controle e comunicação servindo, por exemplo, para habilitar ou não a operação do sistema de navegação durante o voo, já que é permitido a um operador em terra assumir o comando do VANT.

Como saída, o sistema de navegação, de modo geral, produz uma série temporal contendo:

- comandos básicos de navegação, compostos por apenas dois valores: ângulo de rolagem (giro em torno do eixo longitudinal) e altitude. Trata-se de metas que o sistema de navegação continuamente atualiza e informa ao Sistema de Controle, o qual se responsabiliza por fazer o VANT atingi-las o mais rápido possível;
- alarmes para acionamento de periféricos, como câmeras.

Uma vez que o sistema de navegação é parte essencial de um VANT, o estudo e o conhecimento desse sistema faz-se necessário para que o modelo de arquitetura aqui proposto possa ser equacionado da forma mais completa possível.

2.9 Sistema de Controle

Um sistema de controle é composto da interconexão de componentes, seja conectados ou relacionados, de maneira a comandar, controlar ou ajustar a si mesmo ou outro sistema. No caso de VANTs o sistema de controle é responsável por manter a aeronave estabilizada fazendo uso dos servomotores e dos atuadores presentes na aeronave ([OPENPILOT, 2010](#); [ARDUPILOT, 2011](#); [TOEPKE, 2012](#); [NONAMI *et al.*, 2010](#)).

O sistema de controle de um VANT possui, geralmente, três funções principais:

- ativar o modo de voo adequado, que pode ser:

- **modo direto:** os comandos de solo de pilotagem são simplesmente repassados aos atuadores. Dessa forma, o piloto em solo possui controle total sobre a aeronave;
 - **modo estabilizado:** o piloto em solo comanda a aeronave de forma simplificada, com seus comandos sendo modificados antes de serem enviados para os atuadores. Isso permite um controle em tempo real simplificado para pilotos inexperientes;
 - **modo autônomo:** o sistema de controle executa os comandos básicos fornecidos pelo sistema de navegação. Esse modo permite a execução autônoma de um plano de voo pré-estabelecido em solo;
- processar comandos de pilotagem e configuração que vêm codificados em modulação, por exemplo, largura de pulso - *Pulse Width Modulation* (PWM), da mesma forma como são enviados para os servomecanismos. Esses comandos precisam ser decodificados para que possam ser tratados pelos algoritmos de controle, na maioria das vezes;
 - assegurar a condição de voo adequada da aeronave dentro de suas especificações (envelope de voo). Os algoritmos de controle devem ser eficientes e bem calibrados de modo a manter o voo estável da aeronave, mesmo em condições adversas;
 - executar comandos básicos fornecidos pelo sistema de navegação. No modo autônomo esses comandos definem a trajetória a ser seguida pela aeronave para a correta execução da missão.

As entradas do sistema de controle são:

- comandos de pilotagem e configuração recebidos a partir de um transmissor em solo. Os comandos de pilotagem correspondem às posições dos atuadores da aeronave conforme comandado pelo piloto em solo. Os comandos de configuração são comandos diversos que podem vir codificados no sinal transmitido do solo;
- dados de sensores informando a situação da aeronave. Exemplos de sensores: unidade inercial (posição: coordenadas geográficas e altitude, velocidade (nos três eixos) e atitude (ângulos de rolagem, arfagem e guinada), unidade barométrica (velocidade aerodinâmica, vertical e altitude-pressão);
- comandos básicos de navegação que correspondem a manobras básicas da aeronave, utilizadas para atingir determinado objetivo pelo sistema de navegação. Esses comandos são gerados e fornecidos em tempo real pelo sistema de navegação de acordo com o plano de voo especificado e a condição atual da aeronave.

As saídas do sistema de controle são simplesmente os comandos dos atuadores da aeronave, que são os servomecanismos das superfícies de controle aerodinâmicas e o controlador de motores elétricos.

Nos modos de voo estabilizado e autônomo, as funções de controle são executadas por meio de algoritmos de controle, ditos controladores, que utilizam dados de sensores como entrada e geram comandos de saída para os atuadores da aeronave.

2.10 Considerações Finais

Este capítulo apresentou os conceitos básicos, características e classificações de Veículos Aéreos Não Tripulados, focando nas especificidades desse sistema. As classificações apresentadas não apresentam um consenso, entretanto é possível observar a partir delas características e parâmetros que são essenciais para o entendimento e o desenvolvimento do modelo de arquitetura apresentado nesta tese de doutorado.

Não existe uma única definição para o sistema de controle a ser empregado nessas aeronaves, nem mesmo dos sistemas de navegação. Entretanto, características comuns puderam ser observadas e agregadas, de modo a permitir que uma abstração fosse feita.

No próximo capítulo são abordados os modelos arquiteturais, as arquiteturas e principalmente as arquiteturas e modelos voltados para veículos aéreos não tripulados, uma vez que esses modelos serviram de base para a concepção do modelo aqui apresentado.

MODELOS ARQUITETURAIS

3.1 Considerações iniciais

Como visto no [Capítulo 2](#), não existe um consenso sobre a classificação de VANTs e tão pouco uma padronização de seus componentes, fato este que dificulta a reutilização de componentes, sejam de hardware ou de software, em projetos de fabricantes diferentes, aumentando a necessidade de certificação de cada novo componente. Um modelo de arquitetura que guie a padronização é uma das possíveis soluções para esse problema, sendo este um dos objetivos desta tese. Este capítulo tem por objetivo elucidar o que é um modelo de arquitetura e qual o estado da arte.

3.2 Definição

Um modelo constitui uma representação de abstrações a partir das quais podem-se avaliar, de forma racional, as propriedades de instâncias deste modelo. Qualquer modelo deve permitir descrever o conjunto de todas as instâncias possíveis do conceito previamente modelado.

Um modelo de referência, por outro lado, não está diretamente amarrado a nenhum padrão, tecnologia ou outro detalhe de implementação concreta. Ele procura oferecer uma semântica comum que pode ser usada de forma não ambígua por e entre implementações diferentes. Nesse sentido, um modelo de referência tem a intenção de oferecer um alto nível de conceitos comuns.

Arquitetura, por si só, é uma estrutura que identifica, define e organiza componentes. O relacionamento e os princípios de projeto dos componentes, funções e interface estabelecidas entre subsistemas também podem ser definidos por uma arquitetura ([GRANT, 2005](#)). Por outro lado, um modelo de referência para uma arquitetura é uma arquitetura onde as entidades, relacionamentos e unidades de informação envolvidos nas interações entre e dentro dos subsistemas

e componentes são definidos e modelados. Em resumo, é um modelo que incorpora o objetivo básico ou a ideia do sistema e pode ser considerado como uma referência para várias finalidades (JUN *et al.*, 2009; REGLI *et al.*, 2009). O termo modelo de arquitetura utilizado neste trabalho reflete exatamente essa última afirmação: incorporar o objetivo básico e as ideias do sistema.

Os termos estilos arquiteturais e arquitetura de referência também são usualmente utilizados como sinônimos (MENDES, 2002). No entanto, alguns autores como Eickelmann e Richardson (1996) afirmam que arquitetura de referência pode ser vista como uma estrutura que provê uma caracterização das funcionalidades dos sistemas de software de um dado domínio de aplicação. Existem arquiteturas de referências para diversos domínios de aplicação, como por exemplo: para avaliação de desempenho (BARBER; HOLT; BAKER, 2002); para aviônicos (BATORY *et al.*, 1995); para ambientes de teste (EICKELMANN; RICHARDSON, 1996); para aplicações distribuídas (SANDKUHL; MESSER, 2000); arquitetura de referência para sistemas de comércio eletrônico (BASS; CLEMENTS; KAZMAN, 2003); arquitetura de referência para servidores Web (PERROCHON, 1995; ZDUN, 2002); Arquitetura de referência para sistemas robóticos (OLIVEIRA, 2015).

Desta forma, faz-se necessário efetuar um levantamento do estado da arte para descobrir trabalhos que apresentem esforços específicos em torno de arquiteturas ou modelos de arquiteturas para SANTS de modo a poder compor um modelo de arquitetura que possa auxiliar na certificação e na padronização quando do desenvolvimento deste tipo de veículo. Uma revisão sistemática foi realizada para identificar o estado da arte da área em que se insere este doutorado (essa revisão é apresentada em detalhes no Apêndice B).

Neste capítulo é apresentada a síntese dos resultados obtidos com a revisão sistemática no tocante aos artigos selecionados que melhor se relacionam com esta tese de doutorado.

3.3 Estado da arte

A revisão sistemática realizada buscou encontrar trabalhos que apresentem características similares à desta tese de doutorado. Entretanto, como nenhum trabalho exatamente igual ao aqui apresentado foi encontrado, esta revisão sistemática discute artigos publicados que apresentam contribuições importantes para cada um dos tópicos de pesquisa deste doutorado. Esta seção apresenta uma síntese dos resultados obtidos a partir dos trabalhos selecionados na última fase da revisão sistemática.

3.3.1 Arquiteturas e modelos para veículos aéreos

O *National Institute of Standards and Technologies* (NIST) provê um modelo de referência para VANTs (NIST, 2002). O modelo de referência foi proposto para especificar as regras militares, usos e comandos de forma compreensível e intuitiva para um comandante humano. A abordagem de arquitetura proposta pelo NIST é diferente da pretendida neste projeto,

focando um nível de abstração mais baixo, onde as camadas especificam o que os componentes encontrados em um SANT devem fazer.

Outro trabalho voltado para arquitetura em SANT propõe a utilização de uma *Local Area Network* (LAN) para interligar os componentes (PASTOR; LOPEZ; ROYO, 2006). Pastor, Lopez e Royo (2006) citam que este modelo é uma boa escolha por ser amplamente difundido, o que facilitaria a conexão entre os diversos componentes de hardware e que, de acordo com a necessidade da missão, pode-se acoplar e desacoplar sensores, câmeras ou outros dispositivos necessários para o cumprimento da missão. Da mesma forma, os protocolos já existentes podem ser utilizados para realizar a comunicação entre os componentes, inclusive fazendo uso de recursos naturalmente encontrados na *Internet* como SOA (*Service Oriented Architectures*). Informações importantes relacionadas à interligação com SOA são utilizadas como base para a concepção e determinação dos serviços e das interfaces das camadas do modelo de arquitetura apresentado nesta tese.

Wu, Ling e Dong (2009), propõem um modelo de protocolo para uso entre os níveis de aplicação de controle de voo e nível de comunicação, e o protocolo é denominado *Time-triggered Controller Area Network* (TTCAN). Este trabalho não abrange toda a arquitetura de SANT, porém demonstra que a divisão em camadas e a especificação de protocolos é possível e auxilia no desenvolvimento de novas propostas de interligação entre os diversos sistemas que compõem o SANT. Esse trabalho norteia e permite identificar que a divisão em camadas é viável, direcionando a definição dos início e fim das camadas do modelo de arquitetura apresentado neste trabalho.

Em Olson e Burns (2005) é proposto um modelo de arquitetura, a fase III do projeto denominado *Manned/Unmanned Common Architecture Program* (MCAP), utilizado pelo Exército norte-americano para uso em VANTs dos tipos *Future Combat Systems* (FCS) - Sistemas de Combate do Futuro e *Command, Control, Communications, Computers, Intelligence, Surveillance and Reconnaissance* (C4ISR) - Comando, Controle, Comunicações, Computadores, Inteligência, Vigilância e Reconhecimento).

A arquitetura MCAP, fase III, é baseada em componentes eletrônicos e software de prateleira - *Commercial off-the-shelf* (COTS) e interfaces de padrões abertos. Os objetivos do desenvolvimento do modelo foi o de definir e desenvolver uma arquitetura capaz de suportar uma série de plataformas de VANTs do Exército norte-americano demonstrando o desempenho do sistema resultante obtido em ambiente de laboratório.

O desenvolvimento desse modelo contou com o estudo de três classes de VANTs: UCAR, MALE e *Extended Range/Multi-Propose* (ERMP), sendo suas aeronaves: o UCAR (um helicóptero de combate não tripulado); o *Shadow 200* (FCS Class III); e *Fire Scout* e A-160. As classes de aeronaves utilizadas bem como os atributos estudados são ilustrados na Figura 4.

Uma das metodologias adotada pelos autores de forma que a arquitetura pudesse atender

Figura 4 – VANTs de interesse para MCAP III

<p>UCAR</p> <p>Integrado a Unidades de Trabalho. Fornecido as Unidade de Ação para missões específicas. Capacidade de cooperar com aeronaves convencionais. Utilização em missões de Ataque, Combate Restrito, Avaliação de Danos de Batalha, Segurança e Reconhecimento Armado. Controlado do solo pelo Sistema de Comando e Controle de Suporte Aéreo do Exército ou Comando, Controle e Comunicações via LDTC.</p>	<p>Fire Scout (Class IV)</p> <p>Integrado a Unidades de Ação. EO/IV/Laser. RAS/IAW Retransmissão de Comunicação quando equipado com LDTC. Atualmente não utilizado com armamento, mas demonstrações com múltiplas armas estão em andamento. Controlado do solo pelo Sistema de Comando e Controle de Suporte Aéreo do Exército ou Comando, Controle e Comunicações via LDTC. Selecionado inicialmente como veículo de classe IV para Sistemas de Combate do Futuro.</p>
<p>Shadow 200 (Class III)</p> <p>Em plena produção. Sensores EO/IV. Retransmissão de Comunicação. Utilizado pelas Brigadas de Ataque no IRAQUE. Expectativa de ser integrado as Unidades de Ação de Sistemas de Combate do Futuro. Mais espaço. Peso e potência restritos. Utilizado atualmente somente em missões desarmadas. Realiza busca e reconhecimento limitados. Possibilidade de lançamento de pequenas cargas para logística de emergência.</p>	<p>A-160 (ER/MP)</p> <p>Descartável no teatro de operações. Utilizado em missões de longa duração (>24h), sendo ideal para retransmissão de comunicações e vigilância. Normalmente integrado as Unidades de Trabalho. Capaz de trabalhar em conjunto com outros veículos inabitados (aéreos ou terrestres). Pode ser utilizado para lançar sensores autônomos.</p>

Fonte: Adaptada de [Olson e Burns \(2005\)](#).

a todas as plataformas de VANTs selecionadas foi analisar e definir os requisitos necessários para que cada classe de VANT pudesse cumprir as missões designadas. Isto foi feito baseado em cenários, sendo estes decompostos a partir da observação das características operacionais das missões e das plataformas.

Foram utilizadas as definições do *Department of Defense Architecture Framework* (DoDAF) para gerar vários padrões de operações a serem realizadas pelo *Unmanned Air Vehicle Common Architecture Prototype* (UCAP) - Protótipo de VANT com Arquitetura Comum). Na [Figura 5](#) é ilustrado um exemplo de diagrama de contexto para o sistema de processador de missão da MCAP III, onde o círculo representa o processador de missão. O conteúdo de cada retângulo é um exemplo representativo dos possíveis subsistemas, tais como links de comunicação de dados, sensores de voo, de armamentos e sensores de *payload* (radares, câmeras, etc.).

A funcionalidade de processamento é provida por múltiplos recursos de processamento genérico utilizando uma rede com topologia *switch fabric* que fornece uma larga banda de comunicação inter-processos seja via *PCI-Express* ou *serial bus*. A rede suporta tráfego baseado em *Internet Protocol* (IP) por meio de encapsulamento/extração para permitir que aplicações executadas nos processadores se comuniquem via Gigabit Ethernet (GbE).

A arquitetura funcional do UCAP é ilustrada na [Figura 6](#), a linha na cor verde representa uma estrutura virtual de rede Gigabit Ethernet, os componentes dos subsistemas, que utilizam

Figura 5 – Diagrama de contexto do UCAP

Fonte: Adaptada de Olson e Burns (2005).

tecnologia IP, são ligados a esta estrutura e o sistema provê um conjunto de funções tais como: processamento, rede e diversas interfaces para serviços básicos (energia, refrigeração, interfaces de software, dentre outros).

Figura 6 – Arquitetura MCAP III/UCAP

Fonte: Adaptada de Olson e Burns (2005).

Segundo os autores essa arquitetura é simples e aplicável a todos os VANTs elencados no projeto. Para comprovar foram conduzidos testes de laboratório com o objetivo de validar a

habilidade da arquitetura do processador de missão e a infraestrutura para serviços futuros que contemplasse os quatro VANTs.

O *Shadow 200* foi escolhido como a plataforma para os testes. Uma missão de reconhecimento foi selecionada, contemplando: vigilância de rota automatizada, região de interesse e precaução autônoma contra intruso (manobra evasiva). Este teste permitiu aos autores concluir que o projeto desenvolvido com arquitetura reduz os custos e garante a interoperabilidade.

O trabalho de [Olson e Burns \(2005\)](#) propõe uma arquitetura comum para concepção de VANTs, porém limitou-se a demonstrar o uso e a funcionalidade da mesma somente no processador de missão. Os testes de laboratório foram executados somente em um tipo de VANT, o de asas fixas, sendo as outras três aeronaves, as de asas rotativas, citadas no trabalho, mas não foram citados os motivos pelos quais não foram testadas. No [Quadro 5](#) são apresentados os requisitos atendidos pela proposta de [Olson e Burns \(2005\)](#).

Quadro 5 – Requisitos atendidos pela MCAP.

Camada / Requisito	Atende
Monitoramento e controle	parcialmente
Navegação e serviços	parcialmente
Missão	completamente

Fonte: Dados da pesquisa.

O trabalho proposto em [Pastor, Lopez e Royo \(2007\)](#) baseia-se em uma arquitetura de hardware e software desenhada para operar o controlador de missão e *payload* de mini/micro VANTs. Segundo os autores a inovação da proposta está no uso de uma arquitetura de hardware distribuída facilmente escalável pelo uso de LAN, arquitetura de software baseada em subscrição de serviços, abstração da camada de comunicação e fluxo de execução baseado no planejamento de missão.

Na visão do autores um VANT é um sistema complexo, composto por seis módulos principais, os quais são ilustrados na [Figura 7](#), sendo propostos os seguintes módulos:

- A aeronave VANT - Uma plataforma simples, leve, aerodinamicamente eficiente e estável. O VANT possui espaço restrito para o sistema de aviônicos;
- O computador de voo - Um sistema computacional desenhado para obter as informações aerodinâmicas do conjuntos de sensores, interpretá-los e enviar sinais para os componentes que movimentam as superfícies de controle da aeronave, de maneira que esta siga o plano de voo;
- O *payload* - Um conjunto de sensores (câmeras de TV, sensores infravermelhos, sensores térmicos e outros) que captam informações que podem ser parcialmente processadas no próprio VANT e transmitidas à estação de controle terrestre;

- O controlador de missão/*payload* - Um sistema computacional responsável pela execução do controle *payload* seguindo o plano de voo de forma a executar a missão designada ao VANT;
- A estação de controle terrestre - Um conjunto de computadores em solo que acompanham o desenrolar da missão, monitoram os sensores da aeronave por meio de sua telemetria e são capazes de interagir com o *payload* ou no plano de voo;
- A infraestrutura de comunicação – Uma composição de sistemas de comunicação tais como rádios modems, comunicação via satélite ou micro-ondas e outros que garantem a comunicação ininterrupta do VANT com a estação de controle terrestre.

Figura 7 – Principais componentes de um sistema VANT

Fonte: Adaptada de [Pastor, Lopez e Royo \(2007\)](#).

A arquitetura proposta, denominada Computador de Controle de Missão - *Mission Control Computer* (MCC) é construída com um conjunto de microprocessadores embarcados ligados por uma LAN, sendo uma arquitetura distribuída e escalável. O alto nível de modularidade oferecido por uma LAN provê flexibilidade para acoplar o tipo de microprocessador mais adequado ao uso do módulo, atendendo seus requisitos funcionais. Essa arquitetura é ilustrada na [Figura 8](#), sendo que o retângulo laranja representa a infraestrutura LAN/Ethernet, onde os sistemas, na cor azul, como vídeo, controle de motor, armazenamento, comunicação e controle de voo se conectam.

Acima da infraestrutura da LAN foi implementado um nível de software que permite cada módulo computacional suportar múltiplas aplicações. Cada aplicação é criada e registrada como um serviço, que pode ser descoberto e consumido, idêntico ao conceito de *Service Oriented Architectures* (SOA).

Figura 8 – Visão Geral da arquitetura do Computador de Controle de Missão

Fonte: Adaptada de [Pastor, Lopez e Royo \(2007\)](#).

A implementação dos *links* de comunicação sobre a LAN foram possíveis com a utilização de um *gateway* que permite selecionar entre o melhor tipo de *link* de comunicação: os de rádio frequência, micro-onda, via satélite ou outro disponível. A seleção da melhor forma de comunicação em um dado momento da execução do plano de voo é definido levando-se em consideração a largura da banda, a latência e consumo de energia. Na [Figura 9](#) é ilustrada a arquitetura dos *links* de comunicação, assim como na arquitetura principal o retângulo laranja representa a infraestrutura LAN/Ethernet e os azuis os sistemas nela interligados (telemetria, projeção cartográfica, controle remoto de missão, controle remoto de *payload* e piloto remoto). O *Gateway* de dados está ligado a LAN em uma ponta e aos possíveis meios de comunicação na outra.

Segundo os autores, o projeto aborda os elementos que são críticos para o operação de um VANT. A arquitetura é um sistema distribuído baseado em LAN sendo, portanto, altamente modular e escalável de acordo com a exigências das aplicações. A arquitetura das aplicações seguem os paradigmas utilizados na *Web/Internet*, oferecendo baixa complexidade no desenvolvimento e

Figura 9 – Arquitetura de comunicação entre o VANT e estação de controle terrestre

Fonte: Adaptada de [Pastor, Lopez e Royo \(2007\)](#).

inúmeros padrões de protocolos, inclusive alguns com tolerância a falha.

A principal contribuição do projeto proposto por [Pastor, Lopez e Royo \(2007\)](#) é a aplicação de conceitos consolidados, tais como LAN e SOA, na concepção de uma arquitetura. Segundo os autores isto facilita que usuários familiarizados com estes conceitos construam sistemas para SANT. Em contrapartida, a arquitetura baseada em LAN tem como pontos fracos o concentrador ou no caso o *switch* pois uma vez que o mesmo falhe, toda a comunicação entre os componentes cessa. Ademais, o cabeamento necessário para interligação dos componentes tem que seguir os padrões *Ethernet*, ocupam muito espaço e elevam o peso final da aeronave. As camadas presentes neste modelo são ilustradas no [Quadro 6](#).

Em [Deng, Ma e Zhu \(2012\)](#) os autores propuseram um projeto denominado *Reconfigurable Flight Control System Architecture* (RFCSA) ou arquitetura reconfigurável para sistemas de controle de voo. O desenvolvimento do projeto foi realizado seguindo os pré-requisitos:

- **Interface entre módulos simples e unificada** - O módulo funcional deve ser fácil de ser

Quadro 6 – Requisitos atendidos pela MCC.

Camada / Requisito	Atende
Física do segmento aéreo	parcialmente
RTOS distribuída	completamente
Navegação e serviços	completamente
Missão	completamente
Estação de controle terrestre	parcialmente

Fonte: Dados da pesquisa.

adicionado, removido ou substituído para aplicações específicas;

- **Projeto fácil para aplicações de alto nível** - Baixo nível de necessidade de implementar detalhes permite que o usuário desenvolva aplicações de alto nível facilmente;
- **Reutilização de Hardware e Software** - O hardware e software desenvolvidos para uma aplicação podem ser portados para outra sem necessidade de grandes modificações;
- **Baixo custo** - O produto deve ser barato, expandindo o mercado de pequenos VANTs;
- **Confiável** - O hardware e software devem ser de fáceis desenvolvimento e verificação.

A RFCSA consiste de uma série de módulos que atendem aos cinco requisitos citados anteriormente. Todos os módulos são lógicos e conectados via interface de módulos. A interface de módulo é unificada. O modelo é ilustrado na [Figura 10](#), onde cada módulo específico (Processador, Sensor, Atuador, GPS, INS, Comunicação e Energia) possui uma interface que possibilita sua interconexão com os demais.

Segundo os autores, a arquitetura de software desenvolvida para obter as vantagens do sistema modular da RFCSA provê métodos capazes de coordenar o comportamento de todos os módulos, sendo que estes devem ser robustos, tolerantes a falha e flexíveis. A metodologia indicada para resolver problemas de coordenação dos módulos neste sistema distribuído, é o uso de SOA. O conceito de SOA permite a interoperabilidade, compartilhamento e reúso. O sistema torna-se mais flexível e novas funcionalidades podem ser facilmente integradas de acordo com a necessidade do usuário. Nesse modelo de arquitetura de software, o aplicativo, que é executado no módulo Processador, envia Requisições de Serviços aos Módulos de Serviços, como ilustrado na [Figura 11](#).

Os autores, [Deng, Ma e Zhu \(2012\)](#), concluem que o conceito sistemas modulares podem ser aplicados em sistemas aviônicos. Isso é bastante interessante para o trabalho que foi desenvolvido nesta tese, uma vez que sistemas aviônicos modulares permitem sua reconfiguração conforme os requisitos da aplicação. Esse projeto é mais um exemplo do uso de arquitetura modular em VANT, entretanto, sua aplicação ainda é bem modesta, demonstrando um conceito,

Figura 10 – Arquitetura RFCSA

Fonte: Adaptada de [Deng, Ma e Zhu \(2012\)](#).

sem exemplo de aplicação prática. Os requisitos atendidos por essa arquitetura são apresentados no [Quadro 7](#).

Quadro 7 – Requisitos atendidos pela RFCSA.

Camada / Requisito	Atende
Física do segmento aéreo	parcialmente
Abstração do sistema	parcialmente
Monitoramento e controle	parcialmente
Navegação e serviços	parcialmente
Missão	parcialmente

Fonte: Dados da pesquisa.

O projeto apresentado por [Neto et al. \(2012\)](#) trata de uma arquitetura embarcada modular composta de três níveis: sistemas embarcados, *link* de comunicação e sistema de navegação inercial.

O propósito foi projetar e construir uma plataforma de SANT para desenvolvimento e pesquisa de comportamento autônomo. A arquitetura proposta é constituída por componentes eletrônicos embarcados modulares e protocolos de comunicação baseados no modelo *Open System Interconnection* (OSI). Esta arquitetura modular consiste de um circuito principal com um microcontrolador/processador e uma série de módulos necessários para a realização das tarefas, tais como: Unidade inercial e não inercial, unidade de auxílio à navegação, acelerômetros,

Figura 11 – SOA embarcada no RFCSA

Fonte: Adaptada de [Deng, Ma e Zhu \(2012\)](#).

magnetômetro, barômetro, GPS, unidade de controle do motor e outros. A arquitetura modular proposta é ilustrada na [Figura 12](#).

Os autores descreveram sucintamente o que é atendido por cada camada:

- **Camada de hardware:** São os dispositivos de hardware tais como motores, atuadores, sensores, fontes de energia, baterias, equipamentos de comunicação e outros dispositivos que podem ser instalados no VANT.
- **Camada de abstração de hardware:** A interconexão entre os sistemas de hardware foi realizada com barramento de comunicação que utiliza protocolos padrão. Sendo escolhido pelos autores os barramentos RS-232, *Inter-Integrated Circuit* (I2C) e *Serial Peripheral Interface* (SPI) simplificando assim o desenvolvimento do hardware e software.
- **Camada de abstração de sistema:** É definida com um conjunto de elementos de hardware associados para formar um sistema.
- **Camada de serviços:** O *design* do sistema é centrado em serviço, o que permite a interação entre várias partes do VANT que trabalham de forma independente realizando tarefas para atingir o objetivo.

Figura 12 – Arquitetura modular em camadas

Arquitetura Modular em Camadas						
Exemplos de elementos em cada camada da arquitetura						
Camada de Aplicação	Robo Jogador					
	Veículo Aéreo Não Tripulado					
	Veículo Terrestre de Vigilância e Reconhecimento					
Camada de Serviço	Localização e Mapeamento Simultâneos			Monitoramento de Energia	Telemetria	
Camada de Abstração do Sistema	Ativação de Motor	Deteção de Obstáculos	Visão de Computador	Suprimento de Energia	Comunicações	Armazenamento de Dados
Camada de Abstração de Hardware	I ² C					
	SPI					
	RS-232					
Camada de Hardware	Microcontroladores	Conversores A/D e D/A	Dispositivos de I/O	Memórias	Eletrônico de Potência	Dispositivos de Comunicação

Fonte: Adaptada de Neto *et al.* (2012).

- **Camada de aplicação:** Nesta camada as comunicações são realizadas em alto nível, independentemente do meio. Além de acesso a serviços, o usuário é capaz de acessar os elementos de hardware em todas as outras camadas, o que proporciona uma grande flexibilidade no uso dos recursos disponíveis.

Esta proposta é a que mais se assemelha à apresentada nesta tese de doutorado, porém não possui a mesma profundidade do nível de abstração que se pretende atingir. No **Quadro 8** são apresentados os requisitos atendidos por Neto *et al.* (2012).

Quadro 8 – Requisitos atendidos pela LMA.

Camada / Requisito	Atende
Física do segmento aéreo	parcialmente
Abstração do sistema	completamente
Monitoramento e controle	parcialmente
Navegação e serviços	completamente
Missão	completamente
Física do segmento terrestre	parcialmente
Estação de controle terrestre	parcialmente

Fonte: Dados da pesquisa.

3.3.2 A arquitetura IMA (*Integrated Modular Avionics*)

Cabe destaque em detrimento aos outros modelos e arquiteturas o *Integrated Modular Avionics* (IMA) por esta constituir um modelo amplamente utilizado em aeronaves convencionais e por ter apresentado amplo uso na literatura, mais recentemente, para SANTs.

Prisaznuk (1992) propôs o IMA (*Integrated Modular Avionics*), um modelo integrado de aviônicos que é utilizado como uma arquitetura para aeronaves convencionais, sendo a partir de então considerada uma arquitetura neste estudo. A IMA foi proposta inicialmente para ser utilizada na aviação comercial e militar. Ela é definida em torno do conceito de módulos de alto poder de processamento computacional com sistema operacional que permite processamento independente do processamento do software aplicativo. Os módulos de hardware compartilham recursos e são alocados em gabinetes, os quais possuem interfaces bem definidas com a aeronave.

Segundo Watkins e Walter (2007) pode-se diferenciar a arquitetura IMA da arquitetura federada (convencional). Os autores afirmam que a arquitetura IMA fornece compartilhamento de processadores quando do processamento de informações, comunicações e Input/Output (I/O). A fila de recursos é dividida para uso das múltiplas funções de aviônicos. As funções de aviônicos que são atendidas pela IMA podem ser oriundas de diferentes empresas e mesmo assim é garantida sua criticidade devido aos mecanismos robustos de particionamento que são inerentes à arquitetura. Em contraste, arquiteturas federadas de aviônicos implementam coleções independentes de recursos de computação dedicadas (processador, comunicações e I/O) para cada função de aviônicos, normalmente contidos em *Line Replaceable Units* (LRUs) - Unidades de Linha Substituíveis ou *Line Replaceable Modules* (LRMs) - Módulos de Linha Substituíveis. Constituem ainda características da IMA, segundo RIVER (2008): a utilização de padrões abertos e provisão de um único barramento de dados para interligar os principais sistemas da aeronave.

Uma vantagem da IMA em relação à federada é a economia de espaço, peso e consumo de energia ou *Size, Weight, and Power* (SWaP), devido ao fato de uma única unidade executar várias funções. Outra vantagem é a consolidação do hardware, pois tem-se vários aplicativos sendo executados em menos processadores (RIVER, 2008).

A arquitetura IMA foi utilizada por completo no projeto das aeronaves comerciais Boeing 777 e Airbus A380, e na aeronave militar Boeing KC767A. No Quadro 9 é ilustrado um comparativo entre os vantagens e desvantagens da arquitetura federada e não-federada (IMA).

Na Figura 13 é possível observar as diferenças entre arquitetura Federada (a) e IMA (b). Cabe destacar que a Federada necessita de três Central Processing Units (CPUs) enquanto a IMA necessita de apenas uma. De modo análogo, a arquitetura Federada necessita de quatro canais físicos de comunicação enquanto a IMA necessita de somente um.

Após 2010 a arquitetura IMA tem sido aplicada na concepção de SANT. Isso pode ser observado a partir dos trabalhos produzidos (ILARSLAN; BAYRAKCEKEN; ARISOY, 2011). O projeto proposto por Ilarslan, Bayrakceken e Arisoy (2011) baseia-se no desenvolvimento de

Quadro 9 – Comparação entre arquitetura federada e IMA

FEDERADA	IMA
Vantagem	Vantagem
Metodologia Tradicional	SWaP – Várias funções em uma LRU
<i>Design</i> e Certificação - “Fáceis”	Alto grau de reuso de Software
Cadeia de suprimentos consolidada	Alto grau de portabilidade
	Alto grau modular
Desvantagem	Desvantagem
SWaP – Cada função em uma LRU	Poucas aplicações da Metodologia (777, A380)
Baixo grau de reuso de Software	Pouco entendimento
Baixa portabilidade	<i>Design</i> e Certificação - “Complexos”
Baixo grau modular	Cadeia de suprimentos em construção

Fonte: Adaptada de RIVER (2008).

Figura 13 – Exemplo de arquitetura Federada e IMA

(a) Federada

(b) IMA

Fonte: Adaptada de RIVER (2008).

um sistema aviônico para um mini *Vertical Take-Off and Landing* (VTOL) (decolagem e pouso vertical) de propósito acadêmico para estudo da dinâmica de voo utilizando uma arquitetura derivada da IMA.

A construção de um mini UAV é possível, segundo os autores, devido ao surgimento de novas tecnologias como os motores elétricos do tipo *brushless* (ausência de sistema de escovas para alimentação elétrica do induzido) e as baterias de *Lithium Polymer* (Li-Po) que são mais leves se comparadas às baterias de chumbo-ácido ou níquel-metal, mais comuns de serem encontradas. Os objetivos do projeto foram rever os conceitos da arquitetura de sistemas aviônicos e desenvolver algoritmos de controle de voo. Os autores entendem que além dos conceitos básicos da IMA são ainda pré-requisitos desta arquitetura: a aplicação de padrões abertos e uso de módulos comuns hardware e software (desde o barramento de dados até linguagem de software), o uso de equipamento de prateleira - COTS em todos os níveis, o uso de Sistemas Operacionais de Tempo Real - *Real Time Operating Systems* (RTOS) comercialmente

disponíveis que suportem proteção (DO-178C) (RTCA, 2012), segurança *Multiple Independent Levels of Security/Safety* (MILS) e padrões de particionamento (ARINC-653) (RIVER, 2008), e o desenvolvimento de softwares que atendam aos padrões exigidos.

A Figura 14 ilustra a arquitetura proposta pelos autores, que constitui uma modificação da IMA para dar suporte ao VANT por eles proposto. Em um barramento CAN os sensores estão ligados a um computador e a ligação do piloto automático e AHRS a este computador é realizada via RS-232.

Figura 14 – Avionic System Architecture

Fonte: Adaptada de Ilarslan, Bayrakceken e Arisoy (2011).

As principais contribuições desse projeto dizem respeito ao uso de padrões abertos, à preocupação em construir hardware e software que sigam os padrões de certificação e principalmente possibilitar aos envolvidos no projeto se familiarizarem com os princípios de avionica e padrões utilizados na indústria moderna da aviação. Esse trabalho, apesar de apresentar características que se assemelham ao trabalho apresentado neste doutorado não detalha requisitos de sistema de tempo real distribuído, não define as abstrações do sistema, não atende completamente ao sistema de monitoramento e controle, entre outros. Os requisitos atendidos por essa arquitetura são ilustrados no Quadro 10.

Outro projeto que faz uso da arquitetura IMA é o da estação de controle terrestre utilizada no SANT da força área da Turquia conhecido como programa ANKA (KAYAYURT *et al.*, 2011). Este trata-se da estação de controle terrestre utilizada em conjunto com um VANT da categoria MALE que voa a 30.000 pés de altitude - cerca de 9.144m e um raio de alcance de 200 Km. A estação de controle terrestre utiliza dois tipos de softwares: os de tempo real -

Quadro 10 – Requisitos atendidos pelo Mini VTOL.

Camada / Requisito	Atende
Física do segmento aéreo	completamente
RTOS distribuída	parcialmente
Monitoramento e controle	parcialmente
Navegação e serviços	completamente
Missão	completamente
Estação de controle terrestre	completamente

Fonte: Dados da pesquisa.

responsáveis pelo controle de voo, telemetria e interface homem-máquina, que são desenvolvidos em linguagem ANSI C e são executados em dois computadores PowerPC, trabalhando em sistema de redundância tipo Mestre/Escravo e; os de não tempo real - responsáveis pela gravação da telemetria e imagens, teste de pré-voo e outras funcionalidades, que são desenvolvidos com metodologia orientada a objetos e linguagem C# e .Net.

O desenvolvimento de ambos os tipos de software seguem alguns padrões como o uso da OTAN STANAG 4586 - arquitetura geral e mensagem de interfaces para estações de controle terrestre ([NATO/OTAN, 2007](#)); RTCA DO-178B - guia para desenvolvimento de softwares críticos para aviônicos, que define processos, objetivos e atividades para cada nível de criticidade ([PANICKER, 2001](#)); IMA e padrão de desenvolvimento de software particionado ARINC-653 - o software obtém benefícios de particionamento de tempo e espaço.

Segundo os autores, o uso da IMA no projeto de software de tempo real empregado na estação de controle terrestre auxiliou na modularização, proteção e divisão de tempo nos componentes do software. A utilização deste modelo permitiu um desenvolvimento onde cada desenvolvedor era responsável por um módulo e um deles foi responsável pela integração das partes, o que facilitou o desenvolvimento e verificação do produto final. A contribuição em destaque deste projeto é a utilização dos conceitos de IMA no desenvolvimento do software, que estende o conceito aplicado aos aviônicos, para estação de controle terrestre. Observa-se também que o desenvolvimento do software é balizado pela DO-178B que resulta em um software com menos erros e com vistas à certificação. Entretanto o projeto ora apresentado atende quase que exclusivamente o segmento terrestre, como é descrito no [Quadro 11](#), o que o distingue do modelo apresentado nesta tese.

Quadro 11 – Requisitos atendidos pelo projeto ANKA

Camada / Requisito	Atende
Navegação e serviços	parcialmente
Física do segmento terrestre	completamente
Estação de controle terrestre	completamente

Fonte: Dados da pesquisa.

No trabalho apresentado em [Jin et al. \(2012\)](#) os autores optaram pela IMA como alternativa à arquitetura Federada devido ao crescente uso de recursos computacionais nos aviônicos estarem gerando problemas de SWaP.

A arquitetura IMA vem sendo sugerida para resolver os problemas de SWaP e proporcionar melhor consolidação e capacidade de teste do software por meio da modularização em partes. Essa divisão em partes fornece uma maneira eficiente de integrar várias aplicações de tempo real de forma transparente e em um mesmo dispositivo computacional, proporcionando ao mesmo tempo isolamento no ambiente de execução (isso significa que não haverá interrupções por parte de um dispositivo em outro a não ser que isso seja desejado). A ARINC 653 define recursos essenciais para IMA e *Application Programming Interfaces* (APIs) padronizadas ([JIN et al., 2012](#)).

Deste modo, [Jin et al. \(2012\)](#) entendem que o uso dos conceitos da IMA são aplicáveis na construção de um pequeno VANT de baixo custo, permitindo fácil desenvolvimento e extensão, e que permite também a utilização de hardware de prateleira - COTS e software livre que implemente a ARINC 653.

Na [Figura 15](#) é ilustrada a arquitetura ARINC 653 baseada em Linux, proposta por [Jin et al. \(2012\)](#), onde o escalonador de processos, assim como o escalonador de partições são executados em nível de kernel. Cada partição possui sua própria fila de processos e a mudança entre as partições é realizada pela simples troca de ponteiros. O *Health Monitor* é responsável por invocar os manipuladores de erro (*Error Handler*) das partições. A partição *System* contém o gerenciador de inicialização (*Startup Manager*), gerenciador de rede (*Network Manager*) e manipulador de erro, esta partição é executada com alta prioridade. O gerenciador de rede fornece comunicação interpartição transparente sobre diferentes enlaces de rede, que incluem uma *Ethernet Controller Area Network* (CAN) e uma *Serial Bus*. A *ARINC 653 Library* implementa as APIs ARINC 653.

Figura 15 – Arquitetura ARINC 653 baseada em Linux

Fonte: Adaptada de [Jin et al. \(2012\)](#).

O desempenho da implementação foi verificado executando um real *Operational Flight Program* (OFP) para o VANT de asas rotativas denominado GSR-260, em conjunto com um *Video Streaming Program* (VSP) em ambiente de *Hardware in the Loop Simulation* (HILS). Os testes foram feitos visando a implementação de um pequeno VANT com *payload* de cerca de 1 Kg e sua estação de controle terrestre. Devido ao reduzido tamanho do VANT não seria possível embarcar vários dispositivos computacionais capazes de executar o OFP, VSP e outros, sendo a IMA uma alternativa viável para construção deste.

No projeto proposto por Jin *et al.* (2012) demonstra ser viável o uso dos conceitos de IMA para implementar os sistemas embarcados em VANTs, destacando dentre as vantagens a redução de problemas relacionado ao SWaP e a utilização de software livre. Os requisitos apresentados por essa arquitetura é ilustrado no Quadro 12.

Quadro 12 – Requisitos atendidos pelo ARINC

Camada / Requisito	Atende
RTOS distribuída	completamente
Monitoramento e controle	completamente
Navegação e serviços	parcialmente
Missão	completamente
Estação de controle terrestre	parcialmente

Fonte: Dados da pesquisa.

3.4 Considerações finais

Este capítulo apresentou uma revisão bibliográfica sobre arquiteturas, modelos de arquitetura e modelos de referência. O objetivo principal foi a identificação de trabalhos iguais ou muito similares ao desenvolvido nesta tese de doutorado. Como nenhum trabalho exatamente igual foi encontrado, a síntese dos resultados apresentada elencou os principais trabalhos e propostas relacionadas às partes fundamentais desta tese de doutorado.

Vários foram os trabalhos apresentados neste capítulo que auxiliaram na definição das razões para a construção de um novo modelo arquitetural para interconexão de SANTs.

Como pode ser observado no Quadro 13, apesar de existirem vários modelos e arquiteturas na literatura aberta, nenhum dos modelos apresentados contempla todos os requisitos para a confecção de um SANT de caráter genérico, ou seja, todos focam na definição e desenvolvimento de SANTs ou VANTs específicos.

Com base no estado da arte apresentado e no Quadro 13, é possível especificar o LARISSA, cujas camadas derivam da necessidade observada de confecção de um modelo de referência para SANTs de propósito geral e com alto grau de abstração, que é apresentado no próximo capítulo.

Quadro 13 – Comparativo entre os trabalhos encontrados na literatura aberta

Camada / Projeto	MCAP	MCC	RFCSA	LMA	Mini VTOL	ANKA	ARINC
Física do segmento aéreo	-	parcial	parcial	parcial	atende	-	-
RTOS distribuída	-	atende	-	-	parcial	-	atende
Abstração do sistema	-	-	parcial	atende	-	-	-
Monitoramento e controle	parcial	-	parcial	parcial	parcial	-	atende
Navegação e serviços	parcial	atende	parcial	atende	atende	parcial	parcial
Missão	atende	atende	parcial	atende	atende	-	atende
Física do segmento terrestre	-	-	-	parcial	-	atende	-
Estação de controle terrestre	-	parcial	-	parcial	atende	atende	parcial

Fonte: Dados da pesquisa.

LARISSA

4.1 Considerações iniciais

O uso crescente dos VANTs deve fazer com que eles se tornem cada vez mais comuns no espaço aéreo brasileiro, seja para uso em aplicações militares ou civis. Neste cenário, as técnicas propostas neste trabalho visam facilitar o desenvolvimento de aplicações automatizadas de VANTs, permitindo a massificação de missões, e ainda, que esses veículos possam ser inseridos e incorporados mais facilmente ao espaço aéreo, contribuindo assim para a sua disseminação. Como anteriormente mencionado, um modelo de arquitetura permite definir não somente o modelo, mas os serviços representados pelo software e os protocolos de comunicação entre os níveis. Assim, com objetivo de prover uma compreensão mais ampla das partes componentes de um sistema de VANTs, este capítulo é dedicado a descrever o modelo de arquitetura em camadas para interconexão de sistemas em SANTs denominado LARISSA.

4.2 O Modelo

LARISSA (*Layered ARchitecture model for Interconnection of Systems in uAs*) é um modelo de arquitetura obtido a partir do levantamento de requisitos efetuado com diferentes especialistas na área de veículos aéreos, tripulados e não tripulados e de informações obtidas a partir de artigos e documentos específicos da área ([Apêndice A](#)).

No LARISSA os componentes de um SANT foram especificados e divididos, com base na composição dos resultados obtidos a partir da revisão sistemática e do comparativo entre as diferentes arquiteturas e modelos existentes na literatura, em **segmento Aéreo**, **segmento Terrestre** e **camada de Abstração do Sistema**.

O segmento aéreo é hierarquicamente composto pela (i) camada Física, (ii) camada Distribuída de RTOS (*Real Time Operating System*), (iii) camada de Monitoramento e Controle,

(iv) camada de Navegação e Serviços e (v) camada de Missão. O segmento Terrestre é, por sua vez, dividido em (i) camada Física e (ii) camada de Estação de Controle Terrestre.

A separação em camadas permite que o sistema seja dividido em subsistemas que podem ter implementações diferentes e auxilia a separação das partes que compõem um sistema embarcado crítico complexo em diferentes níveis de criticidade. Dessa forma, as vantagens proporcionadas pelo modelo de arquitetura em camadas podem ser aplicadas nas seções de baixa criticidade, tornando o desenvolvimento dessas seções mais simples e flexível. Dentre as vantagens proporcionadas, pode-se citar: configuração separada do ambiente, melhoria da reusabilidade e manutenção, maior nível de abstração e de interoperabilidade, interfaces mais interativas entre dispositivos e sistemas de informação, além da facilidade na utilização de componentes fornecidos por pesquisadores e fabricantes de componentes e software utilizados nos SANTS.

Estas camadas podem ser representadas por meio de modelos, os quais tem o propósito de servirem de guias para elaboração dos SANTS, especificando como será a interligação entre os diversos componentes tais como sensores, circuitos de controle, GPS, *payload*, sensores, *links* de comunicação com a estação de controle terrestre, dentre outros.

Em Tecnologia da Informação (TI), uma arquitetura em camadas é utilizada para definir as responsabilidades específicas de cada camada e a interligação entre elas. Baseadas em um modelo de arquitetura, o fabricante de hardware ou o projetista de software podem desenvolver seus produtos sabendo exatamente em que camada ele irá interagir no SANT, quais são os parâmetros de entrada e saída ou qual o tipo de conexão deve ser utilizada.

Um exemplo de modelo utilizado em TI é o modelo de Referência *International Organization for Standardization* (ISO) - OSI que trata da interconexão de sistemas abertos. Segundo [Tanenbaum e Wetherall \(2011\)](#), deve-se aplicar determinados princípios para definir as camadas, sendo que o modelo OSI adotou ([TANENBAUM; WETHERALL, 2011](#)):

1. Uma camada deve ser criada onde houver necessidade de outro grau de abstração;
2. Cada camada deve executar uma função bem definida;
3. A função de cada camada deve ser escolhida tendo em vista a definição de protocolos padronizados;
4. Os limites da camada devem ser escolhidos para reduzir o fluxo de informações transportadas entre as interfaces;
5. O número de camadas deve ser suficientemente grande para que funções distintas não precisem ser desnecessariamente colocadas na mesma camada e suficientemente pequeno para que a arquitetura não se torne difícil de controlar; e

6. Uma arquitetura para ser considerada completa, deve definir o que cada camada pode realizar, especificar os serviços e os protocolos que são utilizados em cada uma.

O modelo de arquitetura LARISSA segue alguns dos princípios do modelo ISO/OSI:

1. **Camadas criadas quando houver necessidade** - as camadas do LARISSA foram criadas com base nos resultados da comparação dos demais modelos e arquiteturas existentes na literatura, sendo assim, o mínimo de camadas necessários foram criados;
2. **Camadas realizam funções bem definidas** - cada uma das camadas do LARISSA executa funções específicas, ou seja, está limitada ao escopo a que foram definidas;
3. **Número suficientemente grande de camadas** - o número de camadas do LARISSA é grande o suficiente para englobar todos os elementos que compõem o SANT, não se restringindo somente a elementos do sistema de controle ou navegação, sendo assim, um modelo de arquitetura que permite a especificação de diferentes tipos de VANTs;
4. **Arquitetura completa** - o LARISSA é um modelo de arquitetura completo especificando os elementos.

No caso do LARISSA não há definições de protocolos específicos entre camadas, uma vez que, em sendo camadas que representam elementos de SANTs isso não se aplica. Foi definido neste modelo de arquitetura uma camada de abstração, que permite a interligação das camadas. Mais uma vez, o LARISSA se inspira no modelo ISO/OSI, definindo **o que deve ser feito** e não **como deve ser feito**, deixando o modelo de arquitetura independente de tecnologias.

A [Figura 16](#) ilustra o modelo de arquitetura LARISSA proposto de forma hierárquica. O LARISSA contempla o uso do paradigma SOA em partes de baixa criticidade, fazendo uso das vantagens providas por esse paradigma ([MARCONATO et al., 2014](#)).

O uso de SOA em Sistemas Embarcados pode trazer benefícios adicionais, dissociando configuração de ambiente, oferecendo uma interface mais interativa entre os dispositivos e sistemas de informação, e permitindo o uso de serviços de recursos fornecidos por servidores de Internet mais poderosos. Na prática, SOA é capaz de proporcionar a integração dos serviços de sistema de baixo nível, como informações do sensor, e os serviços de sistemas de informação de alto nível, como a gestão de missão. Com base nisso [Rodrigues et al. \(2016\)](#) propuseram um *framework* com objetivo de mitigar as ameaças de utilização de SOA em VANTs. Um dos principais elementos elencados no *framework* é o LARISSA, que facilita a integração e reutilização (proporcionando modularidade e facilidade de manutenção) das várias partes que compõem o SANT. O LARISSA provê toda a infraestrutura para o uso de SOA, entretanto, a forma como SOA é utilizado, o uso de um paradigma não-determinístico em um sistema crítico foge ao escopo deste trabalho e é abordado com detalhes em ([RODRIGUES, 2013](#)).

As seções subsequentes detalham as camadas e subcamadas do LARISSA.

Figura 16 – LARISSA

SANT													
Segmento Terrestre					Camada de Abstração do Sistema	Segmento Aéreo							
Camada Estação de Controle Terrestre	Controle e Monitoramento	Troca de Mapas	Controle de Payload	Vídeo Conferência	IPC	SSI		Controle Automático		Dados Sem Tratamento			Camada de Missão
						Coordenação de Tráfego Aéreo	Controle de Trajetória de Voo	Ciência Geopolítica			Servidor	Camada de Navegação e Serviços	
						Controle de Voo	Tratamento de Emergência	Tratamento de Redundância	Ciência de Aeronavegabilidade	Gerenciamento de Energia	Camada de Monitoramento e Controle		
						API					Camada RTOS Distribuído		
Camada Física	Eletrônicos	Energia	Sistemas Auxiliares	E/S	Driver				Rede			Camada Física	
					Estrutura	Aviônica	Sistemas Auxiliares			Energia			

Fonte: Elaborada pelo autor.

4.3 Camada de Abstração do Sistema

Como já mencionado anteriormente o LARISSA é dividido em dois grandes segmentos, o **Aéreo** e o **Terrestre**, e esses são associados à **Camada de Abstração do Sistema** que permeia entre as camadas com a finalidade de realizar a comunicação entre estas.

A **Camada de Abstração do Sistema**, ilustrada pelo mapa conceitual na [Figura 17](#), é subdividida em duas subcamadas: *Inter-Process Communication* (IPC) e Entrada/Saída (E/S).

É função desta camada definir elementos de abstração que permitam a interoperabilidade entre as camadas, tornando-as independentes de tecnologia. Protocolos diversos podem ser utilizados e acoplados. A subcamada IPC é responsável pela abstração da comunicação entre os processos e a subcamada de E/S pela abstração de funcionamento dos dispositivos de entrada e saída.

O padrão aberto ARINC-653 ([RIVER, 2008](#)) contempla as prerrogativas da arquitetura IMA para particionamento e *safety* determinadas pelo DO-178B ([RTCA, 2012](#)). Uma vez que esse padrão já está validado e é o especificado para uso em SANTs, as subcamadas da Camada de Abstração do Sistema do LARISSA foram estendidas em subcamadas mais específicas de forma a contemplar e estender a ARINC-653 encapsulando o RTOS adotado independente de seu padrão e fornecendo subsídios padronizados para o desenvolvimento de softwares embarcados das camadas superiores.

Figura 17 – Camada Abstração do Sistema

Fonte: Elaborada pelo autor.

Assim, para a subcamada IPC tem-se as subcamadas *Thread* a qual responde pela sua criação e prioridade; *Mutex* para primitivas de exclusão mútua de bloqueio e desbloqueio; *Timer* rotinas que controlam e padronizam o particionamento de tempo; *Semaphore*, controla o acesso aos recursos comuns; *Signal* auxilia na sincronização de *threads*. As subcamadas que compõem a de E/S são: *Driver* para um nível mais detalhado do hardware; *Wait* para funções de espera para recursos de E/S que bloqueiam a execução de processos; *Error* executa funções para controle de erros, sendo que estas duas ultimas *Wait* e *Error*, também estão presentes na IPC.

4.4 Segmento Aéreo

No segmento **Aéreo** tem-se as camadas: **Missão, Navegação e Serviços, Monitoramento e Controle, Abstração do Sistema, RTOS Distribuído e Física**. Como são camadas hierarquicamente compostas, cada camada é composta por subcamadas, as quais são descritas nas subseções subsequentes.

4.4.1 Camada Física do segmento Aéreo

A camada Física do segmento Aéreo, ilustrada na [Figura 18](#) é a camada de hardware da aeronave que pode ser decomposta nas subcamadas: Estrutura, Aviónica, Energia e Sistemas Auxiliares.

Figura 18 – Camada Física

Fonte: Elaborada pelo autor.

A subcamada de Estrutura representa as categorias de aeronaves utilizadas como VANT

segundo suas características aerodinâmicas, estas subcamadas são ilustradas na [Figura 19](#).

Figura 19 – Camada Física - Subcamada de Estrutura

Fonte: Elaborada pelo autor.

A subcamada Asa Fixa representa as aeronaves que possuem os mesmos componentes destas aeronaves em modelos convencionais, tais como Superfícies de Controle (*aileron*s, *flaps* e componentes móveis do leme de direção e estabilizador de voo). A Fuselagem pode ser entendida como sendo o corpo da aeronave onde são fixadas as Asas e os Estabilizadores de Voo. As Asas são os componentes da aeronave que garantem a sustentação, uma das forças envolvidas no voo. As asas podem variar, quanto ao diedro, perfil e envergadura, sendo que estas variações influenciam na aerodinâmica da aeronave. Os Estabilizadores de Voo são responsáveis por manter a aeronave estável e são formados basicamente pelo leme de direção e estabilizador horizontal ou leme de profundidade.

Na subcamada Asa Voadora estão as aeronaves que possuem o formato de uma asa, não possuindo uma fuselagem por exemplo.

A subcamada Asas Rotativas engloba as aeronaves com as mesmas características de um helicóptero, sendo que as asas na realidade são as pás das hélices. Essas hélices são ligadas à um conjunto denominado rotor que impõe uma série complexa de movimentos às hélices e, além do rotativo, alteram o ângulo de ataque das pás, o que permite a essa categoria de aeronave decolar e pousar na vertical, pairar no ar e voar praticamente em todas as direções.

A subcamada Auto Rotação especifica as aeronaves que possuem a característica singular

das asas que provem sustentação não serem ligadas a um sistema propulsor, de modo que girem livremente. Na sua maioria estas aeronaves contam com um sistema moto propulsor que empurra a aeronave e, a medida que esta adquire velocidade as asas giram gerando sustentação.

A subcamada Balões é reservada às aeronaves que utilizam gases mais leves que o ar para flutuar e propulsores com hélices para prover empuxo. Dirigíveis são exemplos destes tipos de aeronaves.

A subcamada Quadricópteros é destinada às aeronaves que possuem a característica de, na maioria dos casos, possuir quatro motores, sendo estes montados em uma sequência que intercala motores que giram no sentido horário e anti-horário. As variações mais comuns deste tipo de aeronave, possuem de três até oito motores. Assim como os helicópteros sua decolagem e pouso são verticais.

Esta camada assim como as demais, apresenta as principais subcamadas que a representa, entretanto o modelo não é fechado, e permitindo assim que subcamadas possam ser inseridas a medida que novos modelos sejam concebidos, de forma a atender as tecnologias que venham a ser utilizadas no domínio SANT.

Os sistemas eletrônicos da aeronave são descritos na subcamada Aviônica ilustrada na [Figura 20](#).

A subcamada Aviônica é composta por uma série de subcamadas, que por sua vez também possuem níveis mais específicos e hierarquicamente organizados são detalhados a seguir.

Aviônica de Controle - componentes eletrônicos responsáveis pelo **Controle de Voo**, que mantém o voo estabilizado; **Controle de Navegação**, o qual atua em conjunto com o **Controle de Voo** e é responsável por seguir o plano de voo. Os **Subsistemas de Controle** atuam nos demais aviônicas de controle, tais como os de *payload*.

A subcamada Aviônica de Comunicação é subdividida em **Ar/Solo**, onde são descritos os subsistemas responsáveis pela transmissão de dados sobre o controle de tráfego aéreo, telemetria, sensores de dados instalados na aeronave e retransmissão de dados, como se fosse uma estação retransmissora. A outra subdivisão é a **Ar/Ar**, onde são descritos os componentes de comunicação entre aeronaves seja para o coordenação de tráfego aéreo ou coordenação de esquadrão, quando vários VANTs estão atuando no mesmo espaço aéreo sob a coordenação de um destes.

A subcamada Aviônica Antena, descreve as Bandas nas quais pode-se realizar a transmissão de dados tais como *Industrial, Scientific and Medical* (ISM), *Low Earth Orbit* (LEO), *S, Geostationary Orbit* (GEO), *Very High Frequency* (VHF) e *Ultra High Frequency* (UHF), além dos tipos antenas: estacionária ou posicional, estas últimas podem ter suas posições alteradas mecanicamente (mecânica) ou por meios elétricos.

A subcamada Aviônica Infraestrutura define a fiação elétrica e os conectores elétricos/e-letrônicos que serão utilizados na subcamada Aviônica.

Figura 20 – Camada Física - Subcamada Aviônica

Fonte: Elaborada pelo autor.

A subcamada Aviônica Servos Motores e Atuadores especifica os dispositivos eletromecânicos que atuam principalmente nas superfícies de controle da aeronave.

A subcamada Aviônica Sensor é a subcamada responsável por interagir com os sensores e traduzir essas informações para os sistemas de monitoramento, navegação e outros. Compõem esta subcamada:

- Monitoramento e previsão do clima, realizado por sensores específicos;
- Controle de Tráfego Aéreo, o qual pode ser realizado por *Transponder*, *Automatic Depen-*

dent Surveillance-Broadcast (ADS-B) ou Controller–Pilot Data Link Communications (CPDLC);

- Voo, com seus subsistemas: GPS, Altímetro Barométrico, Velocidade Aerodinâmica, Magnetômetro, *Inertial Measurement Unit (IMU)*; *Optical Measurement Unit (OMU)* e Superfícies de Controle sendo observados a Posição e o Torque dos mesmos;
- Motor, sendo os sensores para leitura de RPM, *Cylinder Head Temperature (CHT)*, *Exhaust Gas Temperature (EGT)* e Combustível (leitura do nível).

Os sistemas de fornecimento de energia para o VANT, assim como os sistemas propulsão são tratados pela subcamada de Energia que é ilustrada na [Figura 21](#).

Figura 21 – Camada Física - Subcamada de Energia

Fonte: Elaborada pelo autor.

A subcamada Energia está dividida em três subcamadas:

- **Propulsão**, nesta subcamada são elencadas suas categorias Reação (Turbinas e Turbohélice) e Hélice, que podem ser de combustão interna com motores com ciclos de quatro tempos, dois tempos ou ser um motor de *Wankel*. Nesta subcamada de Hélice tem-se ainda os motores Elétricos;
- **Combustível** tem-se: Gasolina de Aviação, Etanol, Querosene e os Elétricos divididos em Baterias, Células de Combustível e Células Solares;
- **Geração de Energia Embarcada**, VANTs maiores podem conter sistemas próprios para geração de energia para componentes eletrônicos ou mesmo para seus motores, sendo: Gerador, Células Solares e Células de Energia;
- **Fonte de Alimentação e Acondicionamento**, trata do local e forma mais adequada para instalar nos VANTs os sistemas de Energia.

Por fim a camada **Física** conta com uma subcamada de **Sistemas Auxiliares** ilustrada na [Figura 22](#).

Esta subcamada divide-se em **Mecanismos de Pouso** e **Mecanismos de Decolagem**.

- **Mecanismo de Pouso** estão divididos em Rodas para pouso de aeronaves de asa fixa em pista, esqui utilizado pelas aeronaves de asas rotativas. Para aeronaves de asas fixas quando não há a possibilidade de uso de uma pista utiliza-se sistema de *airbag*, rede de recuperação ou paraquedas, dependendo das características de cada aeronave;
- **Mecanismos de Decolagem** vão dos convencionais Rodas para asas fixas e Esqui para asas rotativas, até os não convencionais quando do uso de aeronaves de asas fixas e da impossibilidade de utilizar uma pista para decolagem, sendo a catapulta, uso de foguetes auxiliares e veículos de lançamento.

Uma vez descritas as subcamadas da camada Física a próxima camada é a RTOS Distribuído.

4.4.2 Camada RTOS Distribuído

A camada RTOS Distribuído possui apenas duas subcamadas, as quais não são subdivididas, como ilustrado na [Figura 23](#).

A camada de RTOS Distribuído descreve um conjunto de APIs utilizado pelo sistema operacional de tempo real embarcado na aeronave, seja este utilizado como entrada ou saída para o RTOS. A API é composta pelas subcamadas Informação e Controle do Kernel; Gerenciamento de *Thread*; Funções de Espera Genéricas; Gerenciamento de *Timer*; Comunicação Inter *Thread*; Compartilhamento de Recursos e suas subcamadas; Tipos e Definições de Dados Genéricos; *Status* e Códigos de Erros.

Figura 22 – Camada Física - Subcamada de Sistemas Auxiliares

Fonte: Elaborada pelo autor.

4.4.3 Camada de Monitoramento e Controle

A camada Monitoramento e Controle, ilustrada na [Figura 24](#), é responsável pelo monitoramento das ações da aeronave.

Esta camada está dividida nas subcamadas de **Controle de Voo**, **Subcamada de Tratamento de Emergência**, subcamada de **Tratamento de Redundância**, subcamada de **Ciência de Aeronavegabilidade** e subcamada de **Gerenciamento de Energia**. A subcamada de Controle de Voo responde pelos **Comandos Básicos** executados pela aeronave, a **Decolagem Automática** e também o **Pouso Automático**. Já a subcamada de Tratamento de Emergência responde pelos eventos que não estão planejados, como por exemplo um consumo de bateria além do estimado, inviabilizando o término da missão. A subcamada Tratamento de Redundância gerencia os subsistemas duplicados na aeronave que foram instalados para aumentar a confiabilidade da mesma. A subcamada de Ciência de Aeronavegabilidade é responsável por sensores e detectores embarcados na aeronave cuja finalidade é obter informações que um ser humano tem a habilidade de experimentar, tais como sentir o cheiro de fumaça a bordo, sentir a sensação de trepidação da

Figura 23 – Camada RTOS Distribuído

Fonte: Elaborada pelo autor.

aeronave entre outras. A subcamada de Gerenciamento de Energia responde pelo monitoramento dos níveis de energia consumidos pela aeronave.

4.4.4 Camada de Navegação e Serviços

A camada Navegação e Serviços é composta pelas subcamadas de **Coordenação de Tráfego Aéreo**, **Controle de Trajetória de Voo**, **Ciência Geopolítica** e **Servidor** e é ilustrada na [Figura 25](#).

Esta camada é responsável pela navegação da aeronave, emitindo sinais para que esta realize a trajetória necessária para cumprir a missão. A subcamada de Coordenação de Tráfego Aéreo responde pelo tráfego no espaço aéreo onde a aeronave está operando. A subcamada de Controle de Trajetória de Voo orienta a navegação da aeronave seja para atingir os *Waypoints* ou o *Grid* de coordenadas definidos pela missão. A subcamada de Ciência Geopolítica fica responsável entre outras pelo limite virtual no qual a aeronave deve operar. A subcamada de Servidor contém serviços não prioritários que auxiliam a navegação e o cumprimento da missão,

Figura 24 – Camada Monitoramento e Controle

Fonte: Elaborada pelo autor.

serviços estes que podem ser baseados em *World Wide Web* (www) ou *Data Transfer Mechanism* (DTM).

4.4.5 Camada de Missão

A camada Missão, ilustrada na [Figura 26](#), está subdividida em subcamada de *Smart Sensor Interface* (SSI), subcamada de Controle Automático e subcamada de Dados Sem Tratamento.

A subcamada de SSI é responsável por acessar o *Mission Oriented Sensor Array* (MOSA) (PIRES; CHAVES; BRANCO, 2014; PIRES *et al.*, 2011) e realizar toda a verificação, permitindo averiguar se a aeronave reúne todos os atributos para realizar a missão imposta. A subcamada de Controle Automático responde por aceitar os dados da missão (*Uploading*), enviar os dados coletados (*Downloading*) e iniciar, parar, retomar ou cumprir parte da missão (*Start/Stop/Resume/Step*). A subcamada de Dados Sem Tratamento é responsável simplesmente por enviar dados que não precisam de um tratamento adequado. Corresponde a essa camada, o trabalho de simplesmente garantir que os dados cheguem ao destino pré-definido. Cabe destacar que essa camada foi assim subdividida permitindo que a missão seja tratada em um subsistema (incluindo processador) diferente do sistema de controle (o que ocorre usualmente). Essa camada encerra

Figura 25 – Camada Navegação e Serviços

Fonte: Elaborada pelo autor.

Figura 26 – Camada Missão

Fonte: Elaborada pelo autor.

a sequência de camadas do segmento Aéreo, a seguir são descritas as camadas do segmento Terrestre.

4.5 Segmento Terrestre

O segmento Terrestre é composto pelas camadas **Estação de Controle Terrestre** e **Física**. Assim como no segmento Aéreo, cada camada é composta por subníveis que são detalhados nas subseções subsequentes.

4.5.1 Camada Física do segmento Terrestre

A camada Física é ilustrada na [Figura 27](#) e se assemelha em alguns aspectos à camada Física do segmento Aéreo. Entretanto, os elementos apresentados são diferentes em sua composição. A divisão em subcamadas é assim apresentada: **Eletrônicos**, de **Energia** e de **Sistemas Auxiliares**.

Figura 27 – Camada Física

Fonte: Elaborada pelo autor.

A subcamada de Eletrônicos é composta pelos subsistemas: Hardware da Estação, basicamente computadores com CPUs, Monitores de Vídeo, Teclados, *Mouse* e *Joystick* (podem ainda compor esse subsistema mesas tangíveis ou outros tipos de tecnologias); subsistema de Comunicações, idêntico a subcamada no segmento Aéreo. O mesmo conceito é válido para o subsistema de Antena. O subsistema de Sensor possui apenas as funções de Monitoramento e Previsão do Tempo.

A subcamada de Energia é responsável por fornecer energia para o segmento terrestre, possui certas semelhanças com a mesma subcamada do segmento Aéreo, excluindo-se o sistema de propulsão e acrescido das Baterias.

A subcamada de Sistemas Auxiliares também guarda algumas semelhanças ao seu análogo no segmento Aéreo, porém é mais simples nos subsistemas de Mecanismos de Pouso que conta apenas com a Rede de Recuperação e o Mecanismo de Decolagem que conta apenas com a Catapulta e o Veículo de Lançamento. O subsistema de Abrigo é um local onde monta-se a estrutura da estação de controle terrestre. O Suporte à Vida responde pelo suprimento de água e manutenção da temperatura do abrigo, possibilitando condições de trabalho aos operadores do VANT.

4.5.2 Camada Estação de Controle Terrestre

A Estação de Controle Terrestre, ilustrada na [Figura 28](#), possui as subcamadas de Controle e Monitoramento, Troca de Mapas, Controle de *Payload* e Videoconferência.

Figura 28 – Camada Estação de Controle Terrestre

Fonte: Elaborada pelo autor.

A subcamada de Controle e Monitoramento recebe informações da aeronave em forma de telemetria e também pode emitir comandos para guiar a aeronave. A subcamada de Troca de Mapas responde pela capacidade de enviar novos mapas para aeronave quando da alteração da missão inicial. O Controle de *Payload* envia sinais à aeronave com a finalidade de controlar a movimentação e o funcionamento dos sensores, câmeras e radares. A Videoconferência responde pela capacidade de troca de som e imagem com outras estações de controle ou outro local que conte com um sistema deste tipo.

4.6 Especificação do LARISSA

O LARISSA foi especificado em *Unified Modeling Language* (UML), sendo assim, um detalhamento da UML, bem como a metodologia utilizada e os *profiles* gerados por meio da especificação são abordados nas seções que seguem.

4.6.1 UML

Para o entendimento da especificação do LARISSA, conceitos inerentes a UML fazem-se necessários.

A UML surgiu em junho de 1996 e trouxe recursos adequados para modelar cada fase do ciclo de vida do software de maneira estruturada e padronizada. A UML é baseada em técnicas orientadas a objetos e destina-se a visualização, especificação, construção e documentação dos artefatos de um sistema de software (BOOCH; RUMBAUGH; JACOBSON, 2006).

Com a UML é possível modelar qualquer sistema implementado com linguagens orientadas a objetos (Java, C++, *Smalltalk*) e a partir dessa modelagem pode-se, por intermédio

de ferramentas especializadas, gerar código-fonte em uma linguagem alvo. É possível também desenvolver o processo inverso fazendo uso de algumas ferramentas (BLAHA; RUMBAUGH, 2006).

O meta-modelo da UML possui blocos de construção (abstrações, relacionamentos e diagramas) que podem ser estendidos para a criação de um perfil (*Profile*). Um perfil é uma extensão da UML que permite modelar detalhes específicos de um domínio. As abstrações representam detalhes estruturais e comportamentais de um sistema; os relacionamentos declaram como as abstrações se relacionam e diagramas apresentam um resumo do conjunto de abstrações e seus relacionamentos (OMG, 2014).

A linguagem de modelagem UML é a consolidação de outras técnicas de modelagem descritas por Booch, Rumbaugh e Jacobson (2006) e possui alguns mecanismos de extensão que permitem definir conjuntos de elementos notacionais específicos de um determinado domínio, ferramenta ou projeto. Os mecanismos de extensão fornecidos pela UML são os estereótipos (*Stereotypes*), as etiquetas valoradas (*Tagged Values*) e as restrições (*Constraints*). Os estereótipos são usados para estender o vocabulário da UML introduzindo novos tipos de elementos. Cada estereótipo é um novo meta-elemento que estende alguma classe base (*Base class*) do meta-modelo da UML e que pode ser instanciado em algum modelo de projeto. Um estereótipo pode possuir um conjunto de propriedades (etiquetas valoradas), que geralmente são materializadas na forma de atributos. Uma etiqueta é basicamente um par que consiste de um nome e um valor e são representadas entre chaves, por exemplo: {*interface = nameOfInterface*}. As restrições são condições que devem ser mantidas verdadeiras para que a semântica do estereótipo fique correta.

Essas extensões podem ser definidas e agrupadas em um determinado perfil. Perfis visam fornecer elementos de notação, específicos de determinados domínios de aplicação, com o objetivo de melhor representar suas particularidades e assim melhorar o entendimento da especificação. A própria *Object Management Group* (OMG) possui várias iniciativas para a criação de perfis UML (OMG, 2014).

4.6.2 Metodologia utilizada para desenvolvimento do profile LARISSA

Com vistas ao exposto sobre UML, uma extensão da mesma foi desenvolvida e denominada *profile* LARISSA. Esse *profile* utilizou os conceitos da UML2 da OMG (OMG, 2014) para especificar o modelo de arquitetura LARISSA, ou seja o domínio de SANTs, um domínio ainda não especificado.

A ferramenta escolhida para criar o *profile* UML foi o componente *Papyrus*¹ (ECLIPSE, 2014c) da *Model Development Tools* (MDT) do Eclipse² (ECLIPSE, 2014b). O *Papyrus* prove suporte para a *System Modelling Language* (SysML)³ que é uma linguagem de modelagem de

¹ <<https://eclipse.org/papyrus/>>

² <<https://eclipse.org/>>

³ <<https://eclipse.org/papyrus/components/sysml/>>

propósito geral aplicada em engenharia de sistemas suportando especificação, análise, *design*, verificação e validação de sistemas, inclusive sistemas embarcados e sistemas de tempo real (OMG, 2015). Desse modo, é possível verificar que esse conjunto de ferramentas permite a modelagem do *profile* LARISSA.

O *profile* LARISSA estende seus artefatos do SysML que, por sua vez, contempla o meta-model UML. Atentando-se ainda para os padrões de desenvolvimento de software utilizou-se a metodologia de visões 4+1 (KRUCHTEN, 1995; CHOI; YEOM, 2002) sendo: Visão Lógica (*Logical View*), Visão de Processos (*Process View*), Visão de Implementação (*Implementation View*), Visão de Implantação (*Deployment View*) e Visão de Caso de Uso (*Use Case View*). A metodologia de visões 4+1 foi empregada devido ser utilizada em Engenharia de Software para descrever a arquitetura de sistemas com base no uso de múltiplas visões simultâneas.

Para cada visão existe pelo menos um diagrama UML que a representa, assim, definiu-se quais os diagramas UML a serem utilizados para representar cada visão da seguinte forma:

- Visão de Caso de Uso - Diagrama de Caso de Uso (*Use Case Diagram*);
- Visão Lógica - Diagrama SysML de Definição de Bloco (*SysML Block Definition Diagram*);
- Visão de Processos - Diagrama de Sequência (*Sequence Diagram*);
- Visão de Implementação - Diagrama de Componente (*Component Diagram*) - em alguns casos onde necessitou um detalhamento maior, foi utilizado o Diagrama SysML Interno do Bloco (*SysML Block Internal Diagram*) e
- Visão de Implantação - Diagrama de Implantação (*Deployment Diagram*).

Como parte da metodologia, cada uma das visões foram especificadas, detalhadas e exemplos de cada uma dessas visões são aqui apresentados com objetivo de ilustrar e apresentar cada uma delas.

A *Use Case View* é utilizada para realizar o levantamento de requisitos da arquitetura. Na Figura 29 é ilustrado o *Use Case* utilizado, como exemplo, para definição da *SystemBoard*, sendo necessário os requisitos dos sensores (*DefineSensors*), bem como as características físicas de suas conexões, protocolo de comunicação, entre outros; e os requisitos da placa controladora (*DefineBoard*), sendo alguns exemplos o processador, a memória, as interfaces de entrada e saída.

A *Logical View* fornece uma ilustração dos casos de uso, subsistemas, pacotes e classes que auxiliem no entendimento do comportamento do modelo da arquitetura. Na Figura 30 é ilustrada parcialmente a *Logical View* do modelo de arquitetura LARISSA, representando alguns dos subsistemas e pacotes da camada Física, que foram testados para validar o modelo (os resultados de testes são apresentados no próximo capítulo).

Figura 29 – LARISSA Use Case View

Fonte: Elaborada pelo autor.

Figura 30 – LARISSA Logical View

Fonte: Elaborada pelo autor.

O pacote **VANT** representa um modelo que utiliza o *profile* LARISSA, que é dividido em camadas. A camada Física é uma delas e é representada pelo pacote *Physical* (Física), descrito na [subseção 4.4.1](#), que por sua vez é subdividido em várias outras subcamadas. Apesar da extensão, na [Figura 30](#) é ilustrada somente a subcamada *AvionicsEletronics* (Aviônica/Eletrônica) a título de exemplificação desta visão.

Neste ponto cabe um destaque para esclarecer que devido as semelhanças entre as

camadas Aviônicas do segmento Aéreo e Eletrônica do segmento Terrestre, fundiu-se em um único pacote do *profile* estas duas camadas, destacando o reúso de artefatos e simplificando a manutenção do modelo/*profile*. Outro ponto a ser esclarecido, refere-se ao *profile* ser modelado utilizando a língua Inglesa, com a finalidade de facilitar o uso do *profile* em nível global, seja pelos pesquisadores ou pela indústria.

Dentre as várias subcamadas pertencentes a *AvionicsEletronics*, estão representados o *Controller*, *Antenna*, *Infrastructure*, *ServoAndActuator*, *Communications* e *Sensor*. Os pacotes *Communications* e *Sensor* ilustrados na Figura 30 possuem subcamadas como pode ser observado.

A *Process View* é a visão que define os processos que serão implementados. No exemplo da Figura 31 tem-se o procedimento de definição do tipo de placa, e neste exemplo compõe a placa controladora de voo utilizada nos cenários de teste.

Figura 31 – LARISSA Process View

Fonte: Elaborada pelo autor.

A *Implementation View* apresenta uma representação geral do Modelo de Implementação e sua organização em termos de módulos, pacotes e camadas. O diagrama de componentes, ilustrado na Figura 32, representa uma *SystemBoard* do tipo *AllinOne* com seus principais componentes (Sensores).

A *Deployment View* contempla as características físicas do sistema, a conexão entre os subsistemas e a interação hardware e software com objetivo de colocar o sistema em operação. Nesta visão pode-se representar, por meio dos nós, o hardware e a interligação necessária ao funcionamento do sistema. O *SystemBoard*, ilustrado na Figura 33, representa o artefato alvo do

Figura 32 – LARISSA Implementation View

Fonte: Elaborada pelo autor.

teste, sendo que os demais nós representam os componentes do SANT que apresentam interação diretamente.

Figura 33 – LARISSA Deployment View

Fonte: Elaborada pelo autor.

Uma vez apresentadas e ilustradas cada uma das visões, é exemplificado o *profile Controller* (*larissa::UAS::Aerial::Physical::AvionicsElectronics::Controller*). Este *profile* representa

a subcamada de Controle a qual é uma das especificações da subcamada Aviônica, sendo essa uma especificação da camada Física, que pertence ao segmento Aéreo do SANT. Este *profile*, apresentado a título de exemplo, corresponde ao utilizado para modelar a placa de controle de voo, sendo possível escolher o fabricante, modelo e velocidade do microcontrolador a partir de valores pré-definidos. De forma semelhante, define-se o tipo de memória e sua capacidade. Com o *Stereotype Board*, mais precisamente com o atributo *boardType*, define-se qual será a finalidade da placa modelada (*Flight*, *Navigation*, *Subsystem*, *Flight/Navigation*, *Flight/Subsystem*, *Navigation/Subsystem* e *All in One*).

Na Figura 34 é ilustrado o *profile* da subcamada *Flight Sensor* (*larissa::UAS::Aerial::Physical::AvionicsElectronics::Sensor::FlightSensor*), onde estão modelados os principais sensores: *BarometricAltimeter* e seus atributos *altitude*, *temperature* e *pressure*; o sensor *AerodynamicSpeed* e os atributos *sensorConnected*, *sensorValue*, *sensorValueTemperature*, *differentialPressure* e *temperature*.

Figura 34 – *Flight Sensor*

Fonte: Elaborada pelo autor.

O *profile* da UML para o modelo de arquitetura LARISSA foi concebido seguindo um modelo hierárquico, onde os níveis mais altos estão representados por pacotes (*profiles*) e os níveis mais baixos por blocos da SysML em forma de estereótipos. Os pacotes e blocos representam o respectivo módulo da camada do modelo LARISSA, conforme proposto na seção 4.2.

Na [Figura 35](#) é ilustrado o *profile* LARISSA com as camadas em seu nível mais alto, que pode ser considerado equivalente a [Figura 16](#).

Figura 35 – *Profile* LARISSA

Fonte: Elaborada pelo autor.

Uma vez apresentada cada uma das visões é possível apresentar o *profile* LARISSA.

4.6.3 A modelagem do *profile* LARISSA

Ao se modelar um SANT, VANT ou os componentes destes, aplica-se o *profile* LARISSA, o qual possui a seguinte notação: *larissa::UAS::camada*, onde uma camada pode ser composta de subcamadas e, por fim, aplica-se os esterótipos do *profile* escolhido. Os *profiles* e estereótipos modelados para compor o *profile* LARISSA por completo são ilustrados nos quadros desta seção.

No [Quadro 14](#) são ilustrados os dois segmentos no qual o domínio está dividido, *Aerial* - Aéreo e *Ground* - Terrestre que constituem os *profiles* em nível mais alto. Além dos segmentos há uma camada especial que viabiliza a comunicação entre as outras camadas, a *SystemAbstraction*, também constituída como *profile*.

Quadro 14 – Alto nível de abstração do *profile* LARISSA

Nº	Camada	Artefato
0	<i>SystemAbstraction</i>	<i>profile</i>
1	<i>Aerial</i>	<i>profile</i>
2	<i>Ground</i>	<i>profile</i>

Fonte: Elaborada pelo autor.

O *profile SystemAbstraction* e seus estereótipos *IPC* e *IO*, listados no [Quadro 15](#), representam um alto nível de abstração e são utilizados para realizar a comunicação entre as camadas, sendo que o estereótipo *IPC* especifica a comunicação entre processos e o estereótipo *IO* a comunicação com os dispositivos de Entrada e Saída. Neste modelo, a atuação de cada componente é limitada ao modelo de sua própria camada e a comunicação entre as camadas é realizada modelando-se os artefatos necessários utilizando os estereótipos do *profile SystemAbstraction*. Sua notação é *larissa::UAS::SystemAbstraction*.

Quadro 15 – *SystemAbstraction Profile*

Nº	Camada	Artefato
0.1	<i>IPC</i>	Estereótipo
0.2	<i>IO</i>	Estereótipo

Fonte: Elaborada pelo autor.

Seguindo a separação em camadas o [Quadro 16](#) especifica os *profiles* do segmento Aéreo e sua representação, em UML, é *larissa::UAS::Aerial*.

Quadro 16 – *Aerial Profile*

Nº	Camada	Artefato
1.1	<i>Physical</i>	<i>profile</i>
1.2	<i>DistributedRTOS</i>	<i>profile</i>
1.3	<i>MonitoringAndControl</i>	<i>profile</i>
1.4	<i>NavigationAndServices</i>	<i>profile</i>
1.5	<i>Mission</i>	<i>profile</i>

Fonte: Elaborada pelo autor.

A camada *Physical*, é o *profile* obtido a partir da descrição e do modelo descrito na [subseção 4.4.1](#), sendo compostas pelas subcamadas *Structure*, *AvionicsElectronics*, *Energy* e *AuxiliarySystems*, conforme o [Quadro 17](#).

Quadro 17 – *Physical Profile*

Nº	Camada	Artefato
1.1.1	<i>Structure</i>	<i>profile</i>
1.1.2	<i>AvionicsElectronics</i>	<i>profile</i>
1.1.3	<i>Energy</i>	<i>profile</i>
1.1.4	<i>AuxiliarySystems</i>	<i>profile</i>

Fonte: Elaborada pelo autor.

O *profile Structure*, cuja notação é *larissa::UAS::Aerial::Physical::Structure*, possui um estereótipo, *Aircraft*, representado no [Quadro 18](#) onde são representados os principais tipos de aeronaves utilizadas como VANT, sendo a *tagged value classification* específica para este

Quadro 18 – *Structure Profile*

Nº	Camada	Artefato
1.1.1.1	<i>Aircraft</i>	Estereótipo

Fonte: Elaborada pelo autor.

propósito e podendo, no momento da modelagem, receber um dos seguintes valores: *Balloon*, *FixedWing*, *FlyingWing*, *RotaryWing* e *TiltRotor*.

A subcamada Aviônica/Eletrônica é definida no *profile AvionicsElectronics* e este está dividido em outros *profiles*, mais detalhados (Quadro 19).

Quadro 19 – *AvionicsElectronics Profile*

Nº	Camada	Artefato
1.1.2.1	<i>Sensor</i>	<i>profile</i>
1.1.2.2	<i>ServoAndActuador</i>	<i>profile</i>
1.1.2.3	<i>Infrastructure</i>	<i>profile</i>
1.1.2.4	<i>Antenna</i>	<i>profile</i>
1.1.2.5	<i>Communications</i>	<i>profile</i>
1.1.2.6	<i>Controller</i>	<i>profile</i>

Fonte: Elaborada pelo autor.

Cada um dos *profiles* da camada *AvionicsElectronics* é dividido em *profiles* ainda mais específicos e no Quadro 20 são exibidos os *profiles* da subcamada *Sensor*.

Quadro 20 – *Sensor Profile*

Nº	Camada	Artefato
1.1.2.1.1	<i>EngineSensor</i>	<i>profile</i>
1.1.2.1.2	<i>FlightSensor</i>	<i>profile</i>
1.1.2.1.3	<i>AirTrafficControl</i>	<i>profile</i>
1.1.2.1.4	<i>WeatherMonitoringForecasting</i>	<i>profile</i>

Fonte: Elaborada pelo autor.

A notação: *larissa::UAS::Aerial::Physical:: AvionicsElectronics::Sensor::EngineSensor* deve ser utilizada para representar no modelo UML o *profile EngineSensor* (Quadro 21). O estereótipo a ser aplicado é o *EngineSensors* e a *tagged value* mais relevante é a *type*, que pode receber os valores RPM, CHT, EGT, *Fuel*, dependendo do sensor a ser modelado.

Quadro 21 – *EngineSensor Profile*

Nº	Camada	Artefato
1.1.2.1.1.1	<i>EngineSensors</i>	Estereótipo

Fonte: Elaborada pelo autor.

No [Quadro 22](#) são listados os estereótipos do *profile FlightSensor*, cuja notação é *larissa::UAS::Aerial::Physical::AvionicsElectronics::Sensor::FlightSensor*. Estes estereótipos são utilizados para modelar sensores, tais como GPS, Barômetros, Sonares, Acelerômetros, Giroscópios, Bússolas Magnéticas, dentre outros. Dentre as *tagged values* destes estereótipos, destacam-se as que referenciam as posições *X*, *Y* e *Z* de vários dos sensores, como por exemplo GPS, Acelerômetro, Giroscópio e Bússola Magnética. Outras *tagged values* são específicas de alguns sensores, é o caso da *differetialPressure* do estereótipo *AerodynamicSpeed*. O estereótipo IMU, representa o componente que realiza a fusão de dados de dois ou mais sensores.

Quadro 22 – *FlightSensor Profile*

Nº	Camada	Artefato
1.1.2.1.2.1	<i>GPS</i>	Estereótipo
1.1.2.1.2.2	<i>BarometricAltimeter</i>	Estereótipo
1.1.2.1.2.3	<i>SonarAltitude</i>	Estereótipo
1.1.2.1.2.4	<i>UltrasonicRanging</i>	Estereótipo
1.1.2.1.2.5	<i>AerodynamicSpeed</i>	Estereótipo
1.1.2.1.2.6	<i>ControlSurface</i>	Estereótipo
1.1.2.1.2.7	<i>Accelerometer</i>	Estereótipo
1.1.2.1.2.8	<i>Gyroscope</i>	Estereótipo
1.1.2.1.2.9	<i>MagneticCompass</i>	Estereótipo
1.1.2.1.2.10	<i>OMU</i>	Estereótipo
1.1.2.1.2.11	<i>IMU</i>	Estereótipo

Fonte: Elaborada pelo autor.

Uma vez que a apresentação dos demais *profiles* do LARISSA referentes ao segmento Aéreo tornaria o texto muito repetitivo, optou-se por disponibilizar as demais informações no [Apêndice C](#).

O segmento Terrestre, com suas camadas, representam os artefatos que modelam os componentes da Estação de Controle Terrestre e demais componentes necessários para prover o suporte de seu funcionamento e o suporte terrestre necessário para o funcionamento e manutenção dos VANTs. Inicialmente, este segmento está dividido em duas subcamadas, como listado no [Quadro 23](#).

Quadro 23 – *Ground Profile*

Nº	Camada	Artefato
2.1	<i>Physical</i>	<i>profile</i>
2.2	<i>GroundStation</i>	<i>profile</i>

Fonte: Elaborada pelo autor.

A camada Física do segmento Terrestre se assemelha em muito com a camada Física do segmento Aéreo, as principais diferenças são que a do segmento Terrestre possui menos subcamadas, sendo as existentes listadas no [Quadro 24](#) e como citado anteriormente, fundiu-se

a subcamada Aviônica com a Eletrônica, permitindo assim seu uso em ambos os segmentos. Assim, aplica-se aos *profiles AvionicsElectronics, Energy e AuxiliarySystems* as descrições de seus análogos da camada Física do segmento Aéreo e suas subcamadas. A notação utilizada para este segmento é *larissa::UAS::Ground::camada::subcamada*.

Quadro 24 – *Physical Profile*

Nº	Camada	Artefato
2.1.1	<i>AvionicsElectronics</i>	<i>profile</i>
2.1.2	<i>Energy</i>	<i>profile</i>
2.1.3	<i>AuxiliarySystems</i>	<i>profile</i>

Fonte: Elaborada pelo autor.

A camada Estação de Controle Terrestre, foi modelada com o *profile GroundStation*. Os estereótipos listados no [Quadro 25](#) especificam os artefatos desta camada. O estereótipo *ControlAndMonitoring* trata do controle de voo e monitoramento dos dados de telemetria da aeronave, tráfego aéreo e condições climáticas. Foi concebido, um estereótipo específico para controle do *payload*, o *PayloadControl*, devido a necessidade de separação de funções em determinados VANTs, onde um operador pilota a aeronave e outro controla câmeras e radares. O estereótipo *MovingMap* especifica a busca e troca de mapas de terreno com a finalidade de alteração da missão inicial. Para estações de controle terrestre avançadas, onde necessita-se comunicação via vídeo conferência com uma central, tem-se o estereótipo *VideoConference*. A notação para uso destas subcamadas é *larissa::UAS::Ground::GroundStation*.

Quadro 25 – *GroundStation Profile*

Nº	Camada	Artefato
2.2.1	<i>ControlAndMonitoring</i>	Estereótipo
2.2.2	<i>PayloadControl</i>	Estereótipo
2.2.3	<i>MovingMap</i>	Estereótipo
2.2.4	<i>VideoConference</i>	Estereótipo

Fonte: Elaborada pelo autor.

O *profile* LARISSA foi concebido com a finalidade de auxiliar a validação do modelo de arquitetura LARISSA e gerou um produto passível de utilização por pesquisadores e desenvolvedores de SANTS. Este *profile*, sua documentação e vídeos com exemplos, estão disponíveis no site do LSEC⁴.

Uma vez definido, modelado e especificado o *profile* LARISSA é possível vislumbrar e produzir ferramentas que possam auxiliar a confecção, simulação e o desenvolvimento de SANTS que possam ser certificados, e consequentemente, ter sua inserção no espaço aéreo facilitada. A [seção 4.7](#) apresenta uma ferramenta que constitui mais uma contribuição desta tese.

⁴ <http://www.lsec.icmc.usp.br/larissa/>

4.7 Template *larissamodel2simulation*

O *larissamodel2simulation* é um *template* concebido utilizando o Acceleo⁵ que é um gerador de código para uma determinada linguagem alvo, a partir de um modelo UML (ECLIPSE, 2014a). Neste caso o modelo UML utilizado é o *profile* LARISSA.

Um trecho do *larissamodel2simulation* é exibido em Código-fonte 1 e o código completo consta no Apêndice D.

Código-fonte 1: Trecho de código do *larissamodel2simulation*

```

1 [comment encoding = UTF-8 /]
2 /**
3  * The documentation of the module generate.
4  */]
5 [module generate('http://www.eclipse.org/papyrus/0.7.0/SysML', 'http
   ://www.eclipse.org/papyrus/0.7.0/SysML/Blocks', 'http://www.
   eclipse.org/uml2/5.0.0/UML')]
6
7 [import larissamodel2simulation::services::umlservices /]
8
9 /**
10 * The documentation of the template generateElement.
11 * @param aModel
12 */]
13 [template public generateElement(m : Model)]
14 [comment @main/]
15 [for (p:Package | m.eAllContents(Package))]
16     [getPackage( p, 'UAV' )/]
17 [/for]
18 [/template]
19
20 [template public getPackage(p:Package, n:String)]
21     [for (c:NamedElement | p.eAllContents(NamedElement))]
22         [generateClassDeclararions(c)/]
23     [/for]
24 [/template]
25
26 [template public generateClassDeclararions(c:Class) ]
27 [for (p: Property | c.attribute) ]
28     private [p.name/] [p.default/];
29 [/for]
30 [for (es: Stereotype | c.getAppliedStereotypes()) ]
31     [if (es.name = 'RadioModem' )]
32     [file ('uavconf.ini', false, 'UTF-8')]
33 #####

```

⁵ <<https://eclipse.org/acceleo/>>

```

34 #           Parameters for for the channel control           #
35 #####
36     [for ( ps: Property | es.allAttributes() ) ]
37         [if (ps.name = 'frequency')]
38             [if (getTaggedValue(c, es, ps.name) = '_433')]
39 **channelControl.carrierFrequency = 433Hz # - Hz
40         [/if]
41             [if (getTaggedValue(c, es, ps.name) = '_868')]
42 **channelControl.carrierFrequency = 868Hz # - Hz
43         [/if]
44             [if (getTaggedValue(c, es, ps.name) = '_915')]
45 **channelControl.carrierFrequency = 915Hz # - Hz
46         [/if]
47             [if (getTaggedValue(c, es, ps.name) = '_2400')]
48 **channelControl.carrierFrequency = 2.4e9Hz # - Hz
49         [/if]
50             [if (getTaggedValue(c, es, ps.name) = '_5000')]
51 **channelControl.carrierFrequency = 5.0e9Hz # - Hz
52         [/if]
53     [/if]
54     [if (ps.name = 'rfPower')]
55 # max transmission power - mW
56 **channelControl.pMax = [getTaggedValue(c, es, ps.name)]/mW
57     [/if]
58     [if (ps.name = 'sensitivity')]
59 # signal attenuation threshold - dBm
60 **channelControl.sat = [getTaggedValue(c, es, ps.name)]/dBm
61     [/if]
62 [/for]
63 # path loss coefficient alpha
64 **channelControl.alpha = [getTaggedValue(c, es, ps.name)]/
65 **channelControl.numChannels = [getTaggedValue(c, es, ps.name)]/
66 **channelControl.sendDirect = false
67
68 #####
69 #           Parameters for the PHY layer           #
70 #####
71 ##### Phy parameters #####
72     [for ( ps: Property | es.allAttributes() ) ]
73         [if (ps.name = 'rfPower')]
74 **phy.transmitterPower = [getTaggedValue(c, es, ps.name)]/mW
75     [/if]
76     [if (ps.name = 'sensitivity')]
77 **phy.sensitivity = [getTaggedValue(c, es, ps.name)]/dBm
78     [/if]
79     [if (ps.name = 'thermalNoise')]
80 **phy.thermalNoise = [getTaggedValue(c, es, ps.name)]/dBm

```

```

81     [/if]
82     [if (ps.name = 'pathLossAlpha')]
83     **.phy.pathLossAlpha = [getTaggedValue(c, es, ps.name)/]
84     [/if]
85     [if (ps.name = 'snirThreshold')]
86     **.phy.snirThreshold = [getTaggedValue(c, es, ps.name)/] dB
87     [/if]
88     [/for]
89 ##### Phy parameters #####
90     [/file]
91     [/if]
92 [/for]
93 ...

```

O modelo de arquitetura LARISSA é extenso, sendo que algumas camadas são subdivididas em várias outras subcamadas chegando ao nível detalhado do artefato. Isso é importante para permitir que diferentes tipos de VANTs possam ser especificados, fazendo com que o modelo de arquitetura seja abrangente e completo. Com intuito de realizar uma prova de conceito, permitindo a validação do modelo, vislumbrou-se o uso do *profile* para a geração de códigos para um simulador de redes de computadores, uma vez que esse é um dos módulos importantes dentro do LARISSA e estudos relacionados a comunicação em VANTs (inter e intra) são relevantes na área.

Dentre os simuladores de rede de código aberto, ou seja que permitem que o usuário faça alterações em seus módulos ou mesmo crie novos módulos para atender necessidades específicas, destacam-se o NS⁶ (NS, 2014) e o OMNeT++⁷ (OMNET, 2014), ambos simuladores de eventos discretos de redes. O fator que levou a escolha do OMNeT++ e não do NS como simulador utilizado nesta tese foi o fato do OMNeT++ ser um simulador de rede amplamente utilizada na literatura e que permite visualizar e analisar muitos aspectos da rede.

De acordo com Chamberlain (2013), a curva de aprendizagem para utilizar o OMNeT++ é íngreme, principalmente nas etapas iniciais do uso do mesmo. Sendo assim, o *larissamodel2simulation* permite que, em um único ambiente de desenvolvimento sejam integradas todas as ferramentas: o *profile* UML para ser empregado na modelagem do domínio SANT, o gerador de código que extrai os parâmetros do modelo e gera os mesmos automaticamente para o cenário da simulação no OMNeT++. Isso permite que a representação abstrata de um VANT se transforme, em poucos passos, de modelo em simulação, fazendo com que as características do modelo sejam transferidas para a simulação não só minimizando os possíveis erros de uma transcrição manual mas também tornando o processo mais rápido e fácil.

O uso do *profile* LARISSA e do *larissamodel2simulation* com a finalidade de gerar

⁶ <<https://www.nsnam.org/>>

⁷ <<https://omnetpp.org/>>

códigos para o simulador OMNeT++ segue o fluxo ilustrado na [Figura 36](#). A simulação inicia-se com a modelagem em UML das classes que representam o SANT/VANT, em seguida aplica-se o *profile* LARISSA, onde valores são atribuídos as *tagged value*. Posteriormente, caso o modelo seja considerado representativo mas não atenda aos requisitos do *profile* deve ser remodelado. Caso contrário, utilizando o *larissamodel2simulation*, são geradas as configurações dos dispositivos de rede modelados para o projeto do OMNeT++. A simulação é então executada e, se necessário, novos valores são definidos para as *tagged value* e a simulação é reexecutada. Cabe elucidar que as configurações de dispositivos de rede são as geradas uma vez que a camada a ser avaliada é a relacionada a comunicação e o simulador alvo é um simulador de redes.

O mesmo fluxo é seguido quando da necessidade de geração de código para outros simuladores que não de rede. Obviamente a parte do *profile* LARISSA a ser utilizado será o que se relacionar com o simulador correspondente. A título de exemplo pode-se mencionar um simulador de sistema de controle do VANT, ou seja, caso se queira gerar código para um simulador desse tipo cabe selecionar o *profile* com vistas ao simulador para o qual se fará a transformação e, averiguar se ajustes são necessários no *template* do *larissamodel2simulation*. No caso da [Figura 36](#) os blocos *Arquivo de Configuração Gerado*, *Simulação OMNeT++* serão específicos do simulador e da parte do *profile* escolhido.

Figura 36 – Fluxo *profile* LARISSA e *larissamodel2simulation*

Fonte: Elaborada pelo autor.

O desenvolvimento do *larissamodel2simulation* voltado para a parte de redes de computadores foi possível após o estudo de classes que compõem o OMNeT++ e os principais arquivos que fazem parte de cada novo projeto, assim como dos *frameworks* INET (INET, 2014) e o *MiXed siMulator* (MiXiM) (MIXIM, 2014).

4.7.1 Template *larissamodel2simulation* para redes de computadores

Conforme descrito em OMNet (2014), o OMNeT++ é um simulador de evento discreto para o domínio de redes de computadores, orientado a objetos, modular que pode ser programado em C++. Neste simulador é possível modelar redes cabeadas e *wireless*; protocolos; sistemas de hardware distribuído, entre outros.

No OMNeT++:

1. O modelo é constituído por módulos que se comunicam por meio de troca de mensagens;
2. A estrutura do modelo é definida por uma linguagem denominada *Network Description* (NED);
3. Os módulos devem ser programados em C++ utilizando o *kernel* do simulador e suas classes;
4. O arquivo de parâmetros, *omnetpp.ini*, contém a configuração e parâmetros do modelo e pode ser facilmente alterado, alterando assim as características da simulação;
5. Após essas definições compila-se e executa a simulação;
6. Os resultados da simulação são escritos em um arquivo texto e o OMNeT++ possui uma *Integrated Development Environment* (IDE) para gerar gráficos.

Assim é possível observar que o modelo necessário para a geração do cenário de simulação do OMNeT++ baseia-se nos arquivos NED, onde são definidos os módulos utilizados e a organização destes de forma hierárquica. A criação desse modelo pode ser obtida por meio de *frameworks* como MiXiM e INET que oferecem classes de objetos relacionados a redes de computadores mais atualizadas e detalhadas. Sendo assim é necessário conhecer e estudar as classes fornecidas pelo ambiente de simulação. As classes de ambos os *frameworks* foram analisadas para que se pudesse optar por um dos dois. Adotou-se o INET uma vez que ele está em constante desenvolvimento e conseqüentemente é mais atualizado e próximo do mundo real.

Com base no INET os modelos de simulação são concebidos como segue:

- Define-se um arquivo NED que contenha simplesmente o nome do pacote, que por fim será o nome da aplicação gerada;

- Define-se um outro arquivo NED que contenha a especificação do nó, onde será realizada a importação das classes a serem utilizadas relativas aos padrões de rede, mobilidade, aplicação, roteamento, entre outros. Esta tarefa pode ser realizada na IDE do simulador arrastando os ícones referentes a cada módulo em modo *Design* ou escrevendo diretamente o código na linguagem NED em modo *Source*. Na [Figura 37](#) é ilustrado o modo *Design* e o [Código-fonte 2](#) ilustra o modo *Source* de um arquivo NED relativo a um nó.

Código-fonte 2: Exemplo de arquivo NED - modo *Source*

```

1 package larissa_teste;
2
3 import inet.applications.IPingApp;
4 import inet.applications.ITCPApp;
5 import inet.applications.IUDPApp;
6 import inet.base.NotificationBoard;
7 import inet.mobility.static.StationaryMobility;
8 import inet.wpan.linklayer.ieee802154.Ieee802154csmaNic;
9 import inet.networklayer.common.InterfaceTable;
10 import inet.networklayer.ipv4.RoutingTable;
11 import inet.nodes.inet.NetworkLayer;
12 import inet.transport.tcp.TCP;
13 import inet.transport.udp.UDP;
14 import inet.networklayer.IManetRouting;
15 import inet.battery.IBattery;
16 import inet.wpan.Isabel.Base.Display;
17 import inet.linklayer.loopback.LoopbackInterface;
18 import inet.mobility.contract.IMobility;
19
20 module host
21 {
22     parameters:
23         @node();
24         int numTcpApps = default(0);
25         int numUdpApps = default(0);
26         int numPingApps = default(0); // no of PING apps. Specify
the app types in INI file with pingApp[0..1].typename="PingApp"
syntax
27         string tcpAppType = default("");
28         string udpAppType = default("");
29         string routingFile = default("");
30         string routingProtocol @enum("AODVUU","DYMOUM","DYMO","DSRUU",
,"OLSR","OLSR_ETX","DSDV_2","Batman") = default("AODVUU"); //
used mobile routing protocol. see: inet.networklayer.manetrouting
31         string batteryModel @enum("InetSimpleBattery","BasicBattery",
"") = default("InetSimpleBattery"); // used mobile routing
protocol. see: inet.networklayer.manetrouting

```

```

32     string mobilityType = default("StationaryMobility");
33     @display("i=device/uav_s");
34
35     gates:
36         input radioIn @directIn;
37
38     submodules:
39         notificationBoard: NotificationBoard {
40             parameters:
41                 @display("p=60,70;i=block/control");
42         }
43         interfaceTable: InterfaceTable {
44             parameters:
45                 @display("p=60,150;i=block/table");
46         }
47         routingTable: RoutingTable {
48             parameters:
49                 //IPForward = IPForward,
50                 IPForward = true;
51                 routerId = "";
52                 routingFile = routingFile;
53                 @display("p=60,232;i=block/table");
54         }
55         tcpApp[numTcpApps]: <tcpAppType> like ITCPApp {
56             parameters:
57                 @display("p=202,67;i=block/app");
58         }
59         tcp: TCP {
60             parameters:
61                 @display("p=202,150;i=block/wheelbarrow");
62         }
63         udpApp[numUdpApps]: <> like IUdpApp {
64             parameters:
65                 @display("i=block/app;p=347,67");
66         }
67         udp: UDP {
68             parameters:
69                 @display("p=347,150;i=block/transport");
70         }
71         pingApp[numPingApps]: <default("PingApp")> like IPingApp {
72             parameters:
73                 @display("p=450,247");
74         }
75         networkLayer: NetworkLayer {
76             parameters:
77                 proxyARP = false;
78                 arp.globalARP = true;

```

```

79         @display("p=268,247;i=block/fork;q=queue");
80     gates:
81         ifIn[1];
82         ifOut[1];
83     }
84     manetrouting: <routingProtocol> like IManetRouting if
routingProtocol != "" {
85         @display("p=144,247");
86     }
87
88     lo0: LoopbackInterface {
89         @display("p=144,378");
90     }
91
92     // the name of radio interface must have "wlan", valid
examples wlan, wlan802, localwlan ....
93     wlan: Ieee802154csmaNic {
94         parameters:
95             @display("p=347,378;i=block/wrxtx;q=queue");
96     }
97     mobility: <mobilityType> like IMobility {
98         parameters:
99             @display("p=458,67;i=block/cogwheel");
100    }
101    disp: Display {
102        parameters:
103            @display("p=60,378;i=block/app2");
104    }
105    }
106    // battery module
107    battery: <batteryModel> like IBattery if batteryModel != "" {
108        parameters:
109            @display("p=60,308;i=block/plug");
110    }
111    }
112    connections allowunconnected:
113
114        networkLayer.ifOut++ --> lo0.upperLayerIn;
115        lo0.upperLayerOut --> networkLayer.ifIn++;
116
117        for i=0..numTcpApps-1 {
118            tcpApp[i].tcpOut --> tcp.appIn++;
119            tcpApp[i].tcpIn <-- tcp.appOut++;
120        }
121
122        tcp.ipOut --> networkLayer.transportIn++;
123        tcp.ipIn <-- networkLayer.transportOut++;

```

```

124
125     for i=0..numUdpApps-1 {
126         udpApp[i].udpOut --> udp.appIn++;
127         udpApp[i].udpIn <-- udp.appOut++;
128     }
129
130     udp.ipOut --> networkLayer.transportIn++;
131     udp.ipIn <-- networkLayer.transportOut++;
132
133     for i=0..numPingApps-1 {
134         networkLayer.pingOut++ --> pingApp[i].pingIn;
135         networkLayer.pingIn++ <-- pingApp[i].pingOut;
136     }
137     networkLayer.transportOut++ --> manetrouting.from_ip if
routingProtocol != "";
138     networkLayer.transportIn++ <-- manetrouting.to_ip if
routingProtocol != "";
139     // connections to network outside
140     radioIn --> wlan.radioIn;
141     wlan.upperLayerOut --> networkLayer.ifIn++;
142     wlan.upperLayerIn <-- networkLayer.ifOut++;
143 }

```

Um terceiro arquivo NED ainda se faz necessário para completar o ambiente de simulação. Neste caso, o arquivo define a rede, que os *hosts* serão do tipo definido no arquivo NED referente ao nó, a configuração do protocolo IP, obstáculos, se for o caso e quaisquer outras configurações em nível de rede que se fizerem necessárias. Na [Figura 38](#) é ilustrado o arquivo NED em modo *Design* sendo que o [Código-fonte 3](#) representa o mesmo arquivo em modo *Source*, ambos representando a rede.

Código-fonte 3: Network NED - modo *Source*

```

1 package larissa_teste;
2
3 import inet.networklayer.autorouting.ipv4.Ipv4NetworkConfigurator;
4 import inet.world.obstacles.ObstacleControl;
5 import inet.world.annotations.AnnotationManager;
6 import inet.world.radio.ChannelControl;
7
8 network Network
9 {
10     parameters:
11         int numHosts;
12         double nodeSeparation @unit("m") = default(40m);
13     submodules:
14         channelControl: ChannelControl {

```

Figura 37 – Exemplo de arquivo NED - modo *Design*

Fonte: Elaborada pelo autor.

```

15         parameters:
16             @display("p=60,50;i=misc/sun");
17     }
18     configurator: IPv4NetworkConfigurator {
19         config = xml("<config><interface hosts='*' address
= '192.168.0.x' netmask='255.255.0.0' /></config>");
20         @display("p=159,50");
21     }
22     gcs: host {
23         @display("i=device/antennatower_v1;r=,,#707070;p=60,254")
;
24     }
25     uav[numHosts]: host {
26         @display("i=device/uav_1;r=,,#707070;p=295,240");
27     }
28     obstacleControl: ObstacleControl {
29         @display("p=248,50");

```

```

30      }
31      annotations: AnnotationManager {
32          @display("p=350,50;i=msg/paperclip");
33      }
34      connections allowunconnected:
35  }

```

Figura 38 – Network NED - modo *Design*

Fonte: Elaborada pelo autor.

Tendo definido a rede composta pelos *hosts* com todas as suas características, deve-se definir os parâmetros no arquivo *omnetpp.ini*. Estes parâmetros incluem desde a rede a ser executada (em um mesmo modelo de simulação é possível definir mais de uma rede), passando pelo número de *hosts*, tempo de execução da simulação, protocolo de roteamento, protocolo de rede, aplicação, potência de transmissão, sensibilidade de recepção, tipo de mobilidade, nome dos arquivos de saída entre outros, dependendo dos módulos empregados no cenário modelado para simulação. Tais parâmetros podem ser alterados e uma nova simulação executada, por exemplo, para comparar os resultados e definir qual o conjunto de configurações é mais adequado para cada cenário especificado, como é o caso de se querer determinar qual o ponto de equilíbrio entre a potência de transmissão de um dispositivo *wireless* e o seu consumo de bateria. Esta

potência de transmissão permite calcular a que distância um VANT pode voar em relação a outro VANT ou em relação à Estação de Controle Terrestre sem comprometer a transmissão de dados.

O arquivo *omnetpp.ini* é o foco principal do *larissamodel2simulation* para redes de computadores, ou seja, ao modelar um domínio SANT em UML, utilizando o *profile* LARISSA com o auxílio do *larissamodel2simulation*, gera-se parte dos parâmetros automaticamente para o arquivo de configuração do OMNeT++.

No escopo deste trabalho foi implementada por completa a porção que se refere aos componentes dos dispositivos de rede *wireless*, sendo: padrão de comunicação, frequência, potência de transmissão e sensibilidade de recepção. O arquivo gerado é incluído por meio de um comando específico para essa finalidade, no arquivo principal. O [Código-fonte 4](#) é um exemplo de arquivo que o *larissamodel2simulation* gera.

Código-fonte 4: Parâmetros gerados pelo larissamodel2simulation

```

1 #####
2 # Parameters for PHY layer #
3 #####
4 **.phy.transmitterPower = 0.5mW #[mW]
5 **.phy.sensitivity = -95dBm #[dBm]
6 **.phy.thermalNoise = -95dBm #[dBm]
7 **.phy.pathLossAlpha = 2
8 **.phy.snirThreshold = 4dB
9
10 #####
11 # Parameters for the channel control #
12 #####
13 # channel physical parameters
14 *.channelControl.carrierFrequency = 2.4GHz
15 *.channelControl.pMax = 0.5mW
16 *.channelControl.sat = -95dBm
17 *.channelControl.alpha = 2
18 *.channelControl.numChannels = 27
19 #####
20 # Parameters for the display module in the hosts #
21 #####
22 # display parameters (same as channelControl parameters and mac
    parameters)
23 **.disp.carrierFrequency = 2.4GHz
24 **.disp.sat = -95dBm #[dBm]
25 **.disp.alpha = 2
26 **.disp.transmitterPower = 0.5mW #[mW]
27 **.disp.sensitivity = -95dBm #[dBm]
28
29 #####
30 # Parameters for the Energy Model (units: mAh and mA) #

```

```

31 #####
32 **.phy.usage_radio_idle = 1.38mA #[mA]
33 **.phy.usage_radio_recv = 9.6mA #[mA]
34 **.phy.usage_radio_sleep = 0.06mA #[mA]
35
36 **.battery.nominal = 25
37 **.battery.capacity = 25
38 **.battery.voltage = 10
39 **.battery.resolution = 1s
40 **.battery.publishDelta = 0.5
41 **.battery.publishTime = 20s
42
43 **.rxSetupTime = 0.00108s

```

Mais detalhes sobre a lista de classes e a documentação completa do OMNeT++ podem ser obtidas no site⁸.

4.8 Considerações Finais

VANTs são sistemas complexos que realizam missões complexas. VANTs, sejam eles de grande ou pequeno porte, constituem sistemas distribuídos com diversos processadores, componentes e elementos que precisam estar interconectados para que possam funcionar de modo correto. Modelos, arquiteturas e modelos de arquitetura permitem organizar e estruturar esse tipo de sistema. Sendo assim, um modelo de arquitetura baseado em camadas delimita estritamente o que é função de cada camada, bem como os parâmetros que esta recebe da camada logo abaixo e quais parâmetros deve produzir para a camada acima, facilitando assim o desenvolvimento de hardware e softwares, mesmo que por terceiros.

A utilização de um modelo de arquitetura permite a definição de padrões e de formas, principalmente levando à conformidade desses padrões (nesse caso específico de órgãos como a ANAC e a FAA). A divisão em camadas permite reutilizar e atender requisitos, podendo-se inclusive determinar requisitos de certificação ou homologação.

Neste sentido este capítulo apresentou o modelo de arquitetura denominado LARISSA, que tem por objetivo apresentar os componentes de um SANT, bem como os meios de interconexão entre as diferentes camadas em que o modelo é dividido.

Foram apresentadas a modelagem e a especificação do LARISSA que permitem a criação de um SANT do zero, pois o modelo contempla não somente as especificações de todos os sistemas e subsistemas que compõem um SANT, mas permite ainda que o modelo de arquitetura seja estendido para atender novos requisitos. Esta flexibilidade possibilita um fabricante X, por exemplo, projetar e construir a estrutura do VANT, acoplar neste um sistema de controle

⁸ <<https://omnetpp.org/doc/inet/api-current/doxy/annotated.html>>

de missão desenvolvido pelo fabricante *Y* e um sensor desenvolvido pelo fabricante *Z*. Outro benefício previsto é que a longo prazo, o uso de um modelo de arquitetura, pode baixar o custo de projetos de SANT de modo geral, pois possibilita o reuso da tecnologia já desenvolvida, ampliando ainda mais o uso dos VANTs, seja para fins militares ou civis. Isso porque o *profile* LARISSA guia o desenvolvedor na modelagem dos componentes de um SANT e em conjunto com *larissamodel2simulation*, também detalhado neste capítulo, possibilita gerar subsídios que permitem avaliar e validar (por meio de simulação) qualquer parte do modelo de forma rápida e eficaz.

No [Capítulo 5](#) são apresentados os resultados experimentais obtidos com o uso do LARISSA bem como sua validação.

RESULTADOS EXPERIMENTAIS

5.1 Considerações Iniciais

O modelo de arquitetura LARISSA, bem como o *profile* LARISSA e o *larissamodel2simulation* foram validados a partir de experimentos realizados por meio de cenários de simulação, confecção de um simulador e comparação dos resultados obtidos com os de um piloto automático real.

Foram realizados experimentos em quatro cenários que validaram a subcamada Comunicação do segmento aéreo, dois experimentos na camada Controle e um experimento na camada Física (do seguimento aéreo), validando assim o *profile* LARISSA. Além desses experimentos, é também apresentado o *Aerial Vehicle Network Simulator* (AVENS), um simulador de redes de veículos aéreos cujo principal objetivo é fornecer meios de simular a cobertura e a conectividade de redes entre VANTs que voam em cooperação ou que partilham o mesmo espaço aéreo. Resultados experimentais fazendo uso do AVENS também são apresentados.

O LARISSA, como mencionado anteriormente, foi implementado na linguagem de especificação SysML. Os experimentos foram realizados em um computador com processador Intel Core i5-2430M de 2,4 GHz e 6 GB de memória RAM.

A título de ilustração de como foram realizados os experimentos para a validação do LARISSA, foi escolhida a subcamada Comunicação (utilizando o *profile larissa::UAS::Aerial::Physical::AvionicsElectronics::Communications::AirToGround::TelecommandTelemetry*) de modo a detalhar e apresentar os resultados obtidos que corroboram com a hipótese de que modelos de arquitetura auxiliam o desenvolvimento de SANTS/VANTs de forma padronizada facilitando os procedimentos necessários para sua integração no espaço aéreo. O processo de certificação passa a ser facilitado uma vez que é possível fazer uso de ferramentas, como simulação (ANAC, 2010), para certificar algumas etapas do processo.

Cabe afirmar que todas as camadas e subcamadas do modelo de arquitetura LARISSA

foram especificadas e validadas. Entretanto, por falta de espaço e por tornar o trabalho um tanto repetitivo, este capítulo se restringiu a apresentar a validação de camadas e subcamadas cujos resultados permitiram, além da validação do LARISSA, a geração de resultados científicos relevantes por serem inéditos e inovadores.

5.2 Projeto dos experimentos da subcamada Comunicação

A elaboração dos cenários foi iniciada com a modelagem em UML (utilizando o Eclipse Papyrus) e os artefatos que a compõem. A modelagem utilizou a metodologia *4 + 1 views*, como descrito a seguir.

A *Use Case View*, representa em um alto nível os requisitos do modelo ilustrado na [Figura 39](#). Faz-se necessária a definição dos componentes que compõem o SANT¹, sendo esses essenciais para a simulação.

Figura 39 – *Use Case View* dos cenários

Fonte: Elaborada pelo autor.

Na [Figura 40](#) é ilustrada a *Logical View* que representa os pacotes necessários ao VANT. Análoga à esta representação tem-se basicamente os mesmos pacotes aplicados a Estação de Solo, diferenciando apenas os *profiles* aplicados: no VANT, o segmento aéreo enquanto que na Estação de Solo é aplicado o terrestre. No VANT a rede está ligada a uma controladora de voo enquanto na Estação de solo está em um computador. Os pacotes representados são os *profiles* aplicados, possibilitando assim a correta modelagem dos cenários.

Em um nível mais detalhado encontra-se a *Process View* que especifica como deve ocorrer a modelagem dos artefatos de cada um dos cenários, a aplicação dos *profiles* e a composição dos objetos. As mensagens *ChooseBoardType* e *ChooseDevice* indicam que devem ser aplicados estes *profiles* aos respectivos objetos. As mensagens do tipo *Attach* indicam quais artefatos irão compor um objeto. Um exemplo da *Process View* é ilustrada na [Figura 41](#).

¹ Cabe lembrar que a modelagem está toda em inglês, então UAS equivale ao SANT em português. Todas as Figuras apresentam nomes em inglês, mas no texto o mesmo será expresso em português.

Figura 40 – Logical View dos cenários

Fonte: Elaborada pelo autor.

Figura 41 – Process View dos cenários

Fonte: Elaborada pelo autor.

Na definição dos pacotes e blocos, necessários à implementação do modelo UML e representado pela *Implementation View*, são aplicados os *profiles* necessários e definidos os valores das *tagged values*. Para compor os cenários os *profiles* aplicados foram *larissa::UAS::*

Aerial::Physical::AvionicsElectronics::Communications::AirToGround::TelecommandTelemetry, sendo que na Figura 42 é ilustrada a escolha destes *profiles* no processo de modelagem e na Figura 43 é ilustrada a atribuição de valores das *tagged values*; e *larissa::UAS::Aerial::Physical::AvionicsElectronics::Controller*.

Figura 42 – Aplicando o *profile*

Fonte: Elaborada pelo autor.

Figura 43 – Atribuição de Valores a *tagged value*

Fonte: Elaborada pelo autor.

A simulação no OMNeT++ é restrita ao quesito rede uma vez que foca na subcamada Comunicação escolhida para ilustrar o processo. Entretanto é necessário que no modelo UML conste uma *SystemBoard* e *controllerBoard*, necessárias para o correto funcionamento do dispositivo de rede *NetworkDevice*. Nesta etapa da modelagem são definidos os parâmetros iniciais com os quais são configurados os dispositivos de rede do OMNeT++ tais como padrão do dispositivo *wireless*, frequência e potência de transmissão, canal, entre outros. A Figura 44 ilustra esta *view*.

A modelagem do cenário é complementada com a *Deployment View* (Figura 45), representando os artefatos utilizados na implantação.

Com o modelo UML pronto o próximo passo é criar no OMNeT++ um projeto de simulação sendo que a linguagem NED é utilizada para especificar os modelos *host* e *network* a

Figura 44 – *Implementation View* dos cenários

Fonte: Elaborada pelo autor.

Figura 45 – *Deployment View* dos cenários

Fonte: Elaborada pelo autor.

serem simulados.

Com auxílio do *larissamodel2simulation* é gerado o arquivo de configuração utilizando

os valores definidos no modelo UML e utilizados no OMNeT++, um exemplo do uso do *larissamodel2simulation* é ilustrado na Figura 46, onde na caixa de diálogo *Model* é informada a pasta do projeto e nome do modelo UML e, na caixa de diálogo *Target* a pasta do projeto OMNeT++.

Figura 46 – Uso do *larissamodel2simulation*

Fonte: Elaborada pelo autor.

Antes de cada nova simulação, retorna-se ao modelo UML, atribui-se à *tagged value* o novo valor referente ao par Fator/Nível, o arquivo de parâmetros *uavconf.ini* é substituído com o auxílio do *larissamodel2simulation* e a simulação está pronta para ser executada. O processo se repete até todas as simulações serem executadas, sendo o número de cenários, os parâmetros e os *profiles* utilizados todos que se fizerem necessários para o desenvolvedor/fabricante.

Foi dispendida especial atenção no estudo de algumas classes do *framework* INET essenciais para o simulação e consequente validação de parte do LARISSA, destacando-se: *ApplicationBase*, *CSMA*, *MobilityBase*, *UDPBasicApp*.

A conectividade, fornecimento de dados, latência, serviço, mobilidade, natureza 3D, entre outras, são significativamente diferentes em redes de VANTs quando comparadas às redes tradicionais. A maioria dos protocolos e tecnologias *wireless* são desenvolvidas assumindo comunicações 2D, baixa mobilidade (pessoas) ou alta mobilidade (veículos terrestres). Dife-

rentemente, a comunicação de redes de VANTs, conhecidas como *Flying Ad Hoc Network* (FANETs) (BEKMEZCI; SAHINGOZ; TEMEL, 2013), são focadas em comunicação VANT-VANT (A2A) e comunicação VANT-Infraestrutura (A2I), de modo que o grau de mobilidade dos nós é maior do que nas redes tradicionais; a topologia muda frequentemente; necessita de comunicação ponto-a-ponto; o intervalo de comunicação deve ser maior do que o de outras redes; e diferentes tipos de distribuição de dados são necessários.

Uma vez que a simulação é da subcamada Comunicação, a simulação da rede teve que preencher alguns requisitos para que o LARISSA pudesse ser validado (de modo a respeitar características inerentes às FANETS:

1. O módulo físico a ser utilizado, no caso, *Wireless*;
2. O padrão de comunicação: IEEE 802.15.4 e IEEE 802.11;
3. O protocolo de comunicação: *Transmission Control Protocol* (TCP) ou *User Datagram Protocol* (UDP), com a finalidade de gerar os pacotes a serem trafegados;
4. O protocolo de roteamento: nenhum protocolo ou o *Ad hoc On-Demand Distance Vector* (AODV), sendo este o mais comum neste tipo de rede e;
5. O tipo mobilidade dos nodos: movimento em linha reta, aleatório e voo real (seguindo *traces* de voos reais realizados).

Definida a modelagem utilizada, três cenários foram vislumbrados na Tabela 1 e detalhados nas próximas seções.

Tabela 1 – Cenários Avaliados

Cenário	Quantidade de Estação de Controle em Solo	Quantidade de VANTs	Rede
Cenário I	1	1	Enlace
Cenário II	1	2	FANET
Cenário III	1	3	FANET
Cenário IV	1	16 a 128	FANET

5.2.1 Cenário I

O primeiro experimento ilustra um cenário simples (Cenário I) que representa uma situação real em que se tem uma Estação de Controle Terrestre (GCS), um VANT (UAV) e um enlace de dados (*Network*) utilizado seja para a Estação de Controle Terrestre coletar dados de telemetria do VANT ou para enviar dados a ele (como por exemplo um conjunto de *waypoints* a serem seguidos).

Este modelo utiliza o padrão da camada física de rede IEEE 802.15.4, a mobilidade *BonnMotionMobility*, seguindo um conjunto de pontos extraídos de um voo real, a topologia da rede é a ponto-a-ponto e o tráfego de rede é gerado por meio de da aplicação *UDPBasicApp* que utiliza o protocolo UDP.

Foram executadas simulações alterando-se os fatores potência de transmissão e a sensibilidade de recepção dos dispositivos de rede com o intuito de definir a configuração ideal que proporcione a menor perda de pacotes, tornando a rede mais eficiente. A Tabela 2 apresenta os parâmetros utilizados nas simulações. Foram executadas seis simulações, onde o fator potência de transmissão foi executado com os níveis: 1.0 mW, 2.0 mW e 3.0 mW. O fator sensibilidade de recepção foi simulado com os níveis: -95 dBm e -100 dBm.

As simulações combinaram a variação dos níveis de cada fator, ou seja, para potência de transmissão 1.0 mW foi executada uma simulação com a sensibilidade de recepção -95 dBm e outra com -100 dBm, o mesmo ocorreu para os níveis 2.0 mW e 3.0 mW, totalizando assim seis simulações. Cada simulação foi executada somente uma vez, sendo que a documentação² do OMNet++ afirma que em várias execuções da mesma simulação os resultados são sempre os mesmos, uma vez que a semente é sempre a mesma.

Tabela 2 – Parâmetros e valores das simulações do Cenário I

Parâmetros Fixos	Valores					
Frequência	2.4 GHz					
Algoritmo de Roteamento	não se aplica					
Padrão da Camada Física	IEEE 802.15.4					
Aplicação	UDPBasicApp					
Tamanho <i>Playground</i> (x)	2,5 km					
Tamanho <i>Playground</i> (y)	2,5 km					
Tamanho <i>Playground</i> (z)	2,5 km					
Tempo de Simulação	1.000 segundos					
Número de GCS	1					
Tipo de Mobilidade GCS	<i>StationaryMobility</i>					
Número de UAV	1					
Tipo de Mobilidade UAV	<i>BonnMotionMobility</i>					
Taxa de Transferência	250 Kbps					
Pacotes enviados	1.000					
Intervalo médio de envio	1 segundo					
	Simulações Realizadas					
Parâmetros Variáveis	1	2	3	4	5	6
Potência de Transmissão (mW)	1.0		2.0		3.0	
Sensibilidade do Receptor (dBm)	-95	-100	-95	-100	-95	-100

Fonte: Elaborada pelo autor.

Durante o processo de execução da simulação no OMNeT++ pode-se não somente

² <<https://omnetpp.org/doc/omnetpp/manual>>

observar mas também interagir com a mesma por meio do programa de execução que é gerado baseado na linguagem NED. Ao término da simulação um arquivo texto é gerado contendo os resultados da simulação.

Os dados coletados nos arquivos de saída das simulações do OMNeT++ no Cenário I, cujo objetivo foi verificar a comunicação de rede entre o ponto A (VANT) com o ponto B (Estação de Controle Terrestre), obtiveram resultados satisfatórios somente nas simulações onde a potência de transmissão foi de 2 mW e a sensibilidade de recepção -100 dBm e, potência de transmissão 3 mW e sensibilidade de recepção de -100 dBm. Em ambas as simulações não ocorreram perda de pacotes. A perda de pacotes para cada simulação é exibida na [Tabela 3](#). Na [Figura 47](#) é ilustrado o número de pacotes transmitidos (TX) e recebidos (RX), em cada simulação.

Tabela 3 – Percentual de perda de pacotes - Cenário I

Descrição	Valores					
Simulações	1	2	3	4	5	6
Potência de Transmissão (mW)	1.0		2.0		3.0	
Sensibilidade do Receptor (dBm)	-95	-100	-95	-100	-95	-100
% de Perda de Pacotes	96,10	40,00	59,20	0	44,00	0

Fonte: Elaborada pelo autor.

Figura 47 – Pacotes Transmitidos (Tx) e Recebidos (Rx) das Simulações do Cenário I

Fonte: Elaborada pelo autor.

Comparando os resultados, percebe-se que para as simulações de números 4 e 6, com a potência de transmissão em 2 mW e 3 mW, respectivamente, e a sensibilidade de recepção em -100 dBm, não existe perda de pacotes, o que torna ambos os cenários ideais ao que se referem à rede de comunicação de dados. Entretanto ao que se refere aos VANTs é importante observar outro ponto: o consumo de energia. Uma vez que estes veículos carregam consigo uma fonte de energia, na maioria dos casos baterias, que possuem uma determinada capacidade de armazenamento e, o consumo da mesma determina o tempo de duração de uma missão, isto para que o VANT não fique sem um *link* de comunicação de dados com a Estação de Controle Terrestre.

As simulações forneceram dados sobre este aspecto, assim pode-se observar que a potência de 2 mW consumiu 21.749,91 J de energia, contra 22.744,20 J utilizando a potência de 3 mW. Este consumo é ilustrado na [Figura 48](#) onde a linha na cor azul representa o VANT com potência de transmissão 3 mW e a linha amarela o VANT com potência de transmissão 2 mW. As linhas verde e laranja representam a Estação de Controle Terrestre, com 3 mW e 2 mW respectivamente. O consumo de energia possui uma importância maior no VANT quando comparado à Estação de Controle Terrestre, uma vez que esta última encontra-se em solo e pode estar ligada a diversas fontes alternativas de energia, como por exemplo, a rede de energia elétrica, geradores e baterias. O VANT por sua vez conta somente com a fonte de energia nele embarcada. Os resultados sugerem que a melhor configuração é manter potência de transmissão em 2 mW e a sensibilidade em -100 dBm (simulação de número 4) pois não ocorre perda de pacotes de dados e o consumo de energia é menor.

Figura 48 – Consumo de bateria do Cenário I

Fonte: Elaborada pelo autor.

5.2.2 Cenário II

Este experimento retrata um cenário onde VANTs precisam colaborar para execução de uma missão, portanto necessitam manter comunicação não somente com a Estação de Controle Terrestre mas precisam trocar informações entre si. Tem-se desta forma uma Estação de Controle Terrestre (GCS), dois VANTs (UAVs) e uma FANET (Network). Neste cenário avalia-se a comunicação de dados com o uso de um protocolo de roteamento, permitindo assim determinar quando a Estação de Controle Terrestre necessita receber ou enviar dados à um VANT que não está em seu alcance mas existe um segundo VANT que pode ser utilizado como roteador.

O protocolo de roteamento utilizado foi o AODV e os demais dados permanecem os mesmos que do experimento realizado no Cenário I. Foram executadas seis simulações, assim como no Cenário I, permanecendo assim os mesmos fatores e níveis. Os parâmetros utilizados nas simulações, constam da [Tabela 4](#).

Tabela 4 – Parâmetros e valores das simulações do Cenário II

Parâmetros Fixos	Valores					
Frequência	2.4 GHz					
Algoritmo de Roteamento	AODV					
Padrão da Camada Física	IEEE 802.15.4					
Aplicação	UDPBasicApp					
Tamanho <i>Playground</i> (x)	2,5 km					
Tamanho <i>Playground</i> (y)	2,5 km					
Tamanho <i>Playground</i> (z)	2,5 km					
Tempo de Simulação	1.000 segundos					
Número de GCS	1					
Tipo de Mobilidade GCS	<i>StationaryMobility</i>					
Número de UAV	2					
Tipo de Mobilidade UAV	<i>BonnMotionMobility</i>					
Taxa de Transferência	250 Kbps					
Pacotes enviados	1.000					
Intervalo médio de envio	1 segundo					
	Simulações Realizadas					
Parâmetros Variáveis	1	2	3	4	5	6
Potência de Transmissão (mW)	1.0		2.0		3.0	
Sensibilidade do Receptor (dBm)	-95	-100	-95	-100	-95	-100

Fonte: Elaborada pelo autor.

Neste tipo de simulação é possível acompanhar visualmente a ocorrência de falta de comunicação direta entre UAV[1] e a GCS ([Figura 49](#)) e o funcionamento do protocolo de roteamento, uma vez que o UAV[0] recebe o pacote e o retransmite para Estação de Controle Terrestre. Este comportamento é ilustrado na [Figura 50](#).

Os resultados do cenário II, diferem do cenário I, uma vez que o segundo UAV está mais distante da GCS, necessitando que outro UAV realize o trabalho de rotear os pacotes. Após a

Figura 49 – UAV[1] sem comunicação direta com GCS

Fonte: Elaborada pelo autor.

Figura 50 – UAV[0] roteando pacote do UAV[1]

Fonte: Elaborada pelo autor.

coleta e tratamento dos dados das simulações do cenário II, nota-se que somente a simulação de número 6, a qual combina a potência de transmissão em 3 mW e a sensibilidade em -100 dBm não apresentou perda de pacotes. A [Tabela 5](#) detalha a perda de pacotes das demais simulações deste cenário. A quantidade de pacotes enviados (TX) e recebidos (RX) são ilustradas na [Figura 51](#).

Tabela 5 – Percentual de perda de Pacotes - Cenário II

Descrição	Valores					
Simulações	1	2	3	4	5	6
Potência de Transmissão (mW)	1.0		2.0		3.0	
Sensibilidade do Receptor (dBm)	-95	-100	-95	-100	-95	-100
% de Perda de Pacotes	100	67,30	97,70	29,40	73,50	0

Fonte: Elaborada pelo autor.

Como mencionado anteriormente, tem-se somente uma configuração, simulação de

Figura 51 – Pacotes Transmitidos (Tx) e Recebidos (Rx) das Simulações do Cenário II

Fonte: Elaborada pelo autor.

número 6, onde não ocorre perda de pacotes, sendo a eleita como a ideal, para o referido cenário. Outros dados elencados são: o número de pacotes que foram repassados, do UAV[1] passando pelo UAV[0], 960 pacotes e o consumo de energia dos UAV[0], 25.663,49 J e UAV[1] 24.978,20 J. A título ilustrativo na [Figura 52](#) constam também o consumo dos UAVs e GCS com potência de 2 mW.

5.2.3 Cenário III

O terceiro experimento vislumbrou a transmissão de vídeo do VANT para a Estação e Controle Terrestre, uma situação em que se tem uma Estação de Controle Terrestre (GCS), três VANTs (UAVs) e uma FANET (*Network*). Neste cenário avalia-se a comunicação de dados com o uso de um protocolo de roteamento e um padrão de camada física que suporte o volume de dados a ser transmitido. Este cenário simula a Estação de Controle Terrestre recebendo imagens de um VANT que não está a seu alcance sendo necessário utilizar um outro VANT como roteador.

O tráfego de rede é gerado por meio do *UDPVideoStream* e o padrão da camada física de rede é o IEEE 802.11. As demais características do cenário não sofreram alterações. A potência de transmissão assumiu os níveis 20, 25 e 30 mW, e a sensibilidade, os níveis -90 e -92 dBm, diferentemente o que ocorre no padrão 802.15.4, o valor da sensibilidade afeta diretamente a taxa de transferência de dados. Os parâmetros utilizados nas simulações constam da [Tabela 6](#).

Assim como na simulação do cenário II ocorre a falta de comunicação direta entre

Figura 52 – Consumo Bateria do Cenário II

Fonte: Elaborada pelo autor.

Tabela 6 – Parâmetros e valores das Simulações do Cenário III

Parâmetros Fixos	Valores					
Frequência	2.4 GHz					
Algoritmo de Roteamento	AODV					
Padrão da Camada Física	IEEE 802.11					
Aplicação	UDPVideoStream					
Tamanho <i>Playground</i> (x)	2,5 km					
Tamanho <i>Playground</i> (y)	2,5 km					
Tamanho <i>Playground</i> (z)	2,5 km					
Tempo de Simulação	1.000 segundos					
Número de GCS	1					
Tipo de Mobilidade GCS	<i>StationaryMobility</i>					
Número de UAV	3					
Tipo de Mobilidade UAV	<i>BonnMotionMobility</i>					
Taxa de Transferência	9 Mbps (-90 dBm)		6 Mbps (-92 dBm)			
Pacotes enviados	10.486					
Intervalo médio de envio	10 milissegundos					
	Simulações Realizadas					
Parâmetros Variáveis	1	2	3	4	5	6
Potência de Transmissão (mW)	20.0		25.0		30.0	
Sensibilidade do Receptor (dBm)	-90	-92	-90	-92	-90	-92

Fonte: Elaborada pelo autor.

UAV[0] e a GCS (Figura 53) e o roteamento ocorre quando o UAV[2] recebe o pacote e o retransmite para GCS. Este comportamento é ilustrado na Figura 54.

Figura 53 – UAV[0] sem comunicação direta com GCS

Fonte: Elaborada pelo autor.

Os resultados das simulações do cenário III demonstram que não ocorrem perda de pacotes quando do uso de potência de transmissão ajustado para 25 mW e 30 mW e a sensibilidade do receptor em -92 dBm, ou seja, as simulações de números 4 e 6. Para as demais simulações houver perda de pacotes e o percentual destas perdas está detalhado na Tabela 7. A quantidade de pacotes enviados (TX) e recebidos (RX) é ilustrada na Figura 55.

Tabela 7 – Percentual de perda de pacotes - Cenário III

Descrição	Valores					
Simulações	1	2	3	4	5	6
Potência de Transmissão (mW)	20.0		25.0		30.0	
Sensibilidade do Receptor (dBm)	-90	-92	-90	-92	-90	-92
% de Perda de Pacotes	56,52	8,43	38,63	0	16,21	0

Fonte: Elaborada pelo autor.

As simulações apresentaram outros resultados valendo a pena destacar o *delay* que é o tempo de atraso de um bit sair do emissor (TX) até chegar no receptor (RX). Para aplicação simulada este é um fator relevante devido ao atraso durante a transmissão gerar cortes na imagem ou mesmo prejudicar o tempo de resposta para uma ação, como por exemplo o pouso de uma

Figura 54 – UAV[2] roteando pacote do UAV[0]

Fonte: Elaborada pelo autor.

Figura 55 – Pacotes Transmitidos (Tx) e Recebidos (Rx) das Simulações do Cenário III

Fonte: Elaborada pelo autor.

aeronave. Neste sentido a simulação que atingiu o menor *delay* foi a de número 6, com média de 0,004548 segundos. Na Figura 56 são ilustrados os valores médios (em segundos) de *delay* para todas as simulações do cenário III. Levando-se em consideração os resultados sugere-se que a simulação que possui a melhor configuração para o cenário é a de número 6, sendo a potência de transmissão de 30 mW e a sensibilidade de recepção de -92 dBm.

Figura 56 – Delay fim-a-fim das simulações do Cenário III

Fonte: Elaborada pelo autor.

5.2.4 Cenário IV

O objetivo desse experimento foi avaliar qual tipo de topologia é mais eficiente para uma rede FANET, a *mesh* ou a estrela. Para isso foram utilizados o algoritmo de roteamento AODV, o *larissamodel2simulation* e o *profile* LARISSA. Com objetivo não só de responder à questão científica sobre FANETs, mas também o de validar o *profile* LARISSA e o *larissamodel2simulation*, foram realizados experimentos reais para que os resultados obtidos pudessem ser comparados aos da simulação.

5.2.4.1 Experimento Real

Para a realização dos experimentos realizados neste cenário foi utilizado um arquivo de *trace* obtido a partir de experimento real realizado por dois VANTs idênticos de asa fixa (cuja especificação é detalhada no Quadro 26 (MARCONATO *et al.*, 2016)) e uma estação de base³.

Uma das aeronaves (*Drone 1 - DR1*) encontra-se voando a uma distância de 250-500 metros das estação de solo (CGC), enquanto que a segunda (*Drone 2 - DR2*) voa a uma distância

³ Cabe ressaltar que esse trabalho foi realizado em parceria com pesquisadores do *Ecole Nationale de l'Aviation Civile* (ENAC) da França sendo que esses pesquisadores fazem parte do grupo de desenvolvimento do piloto automático Paparazzi.

Quadro 26 – Principais características do VANT DT-18

Característica	Valor
Modelo	DT-18
<i>Payload</i>	250 g
Alcance	100 km
Velocidade de cruzeiro	50 km/h
Vento	acima de 45 km/h
Transmissão em tempo real	acima de 15 km. Extensão para 100 km
Piloto automático	<i>Delair-Tech technology</i>
Computador Embarcado	<i>payload</i> e controle de comunicação, 1 Ghz

Fonte: Adaptada de [Marconato et al. \(2016\)](#).

1500-2000 metros da estação de solo. Desse modo, a distância entre as duas aeronaves é de aproximadamente 1500m. A altitude média das aeronaves é estimada em 1127 pés (345m).

Os dados trocados entre as aeronaves são de cinco tipos diferentes, detalhados no [Quadro 27](#)

Quadro 27 – Diferentes tipos de dados trocados durante o voo

Tipo de Tráfego	Fonte – Destino	Tamanho	Taxa
<i>Heartbeat</i> ou <i>Tick</i>	GCS – Dr1 GCS – Dr2	64 Bytes	1 pacotes/s
Referência Geográfica	Dr1 – GCS Dr2 – GCS	80 Bytes	3 pacotes/s
C2	GCS – Dr1 GCS – Dr2	80 Bytes	1 pacote/s
Vídeo	Dr2 – Dr1 – GCS	1400 Bytes	25 UDP pacotes/segundos width=720,height=576
Rede	Troca entre Dr1, Dr2 e a GCS	Requisição: 66 Bytes Resposta: 62 Bytes HELLO : 62 Bytes Erro: 54 Bytes	1 pacote/s para o HELLO Pacotes de Requisição e Resposta e de Erro são trocados durante a desconexão (perda de rota)

Fonte: Adaptada de [Marconato et al. \(2016\)](#).

Para a comunicação entre as aeronaves e a estação de solo são utilizadas as interfaces Wi-Fi IEEE 802.11n com comunicação *half-duplex* com largura de banda de 5 Mhz.

Os resultados do experimento real realizado são ilustrados no [Quadro 28](#).

Os resultados obtidos com o experimento real permitem avaliar que o uso do AODV obtém um bom resultado quando utilizado com o IEEE 802.11n. A discussão mais detalhada sobre esses resultados pode ser obtida em ([MARCONATO et al., 2016](#)), e a mesma não será aqui relatada uma vez que o objetivo aqui é comparar os resultados obtidos com o experimento real e simulado.

Quadro 28 – Resultados do experimento de uma FANET

Métrica	Valor
Tempo do experimento	30 min
Sobrecarga de pacotes de controle	352 k
Sobrecarga de Tráfego % (bytes)	2,15 %
Tempo médio de vida da rota	14,33 s
Tempo médio de atraso para o re-estabelecimento da rota	0,001098 s
Tempo médio de atraso de transmissão entre a GCS e o DR1	0,000153 s
Tempo médio de atraso de transmissão entre DR1 e DR2	0,000549 s
Conectividade durante os estados de instabilidade	97 %
Número de pacotes de controle perdidos	284 pacotes
Tamanho médio do pacote de carga útil durante a perda de caminho	52 Bytes

Fonte: Adaptada de [Marconato et al. \(2016\)](#).

5.2.4.2 Simulação

Fazendo uso do *profile* LARISSA e do *larissamodel2simulation*, que permitiram a geração de código automática para o OMNeT++ garantindo as especificidades das aeronaves e da estação de controle terrestre, foi possível executar uma série de simulações.

Como anteriormente mencionado, o arquivo de *trace* do experimento real foi utilizado, garantindo que as rotas, distâncias e todos os parâmetros do modelo real fossem incorporados na simulação. Os parâmetros utilizados para a realização da simulação foram:

- **Protocolo de comunicação:** foram utilizados os protocolos IEEE 802.11n (devido ao seu amplo uso em VANTs) e o IEEE 802.15.4 (devido ao seu baixo custo, baixo consumo de energia e alta conectividade);
- **Topologia da rede:** topologia estrela (por ser a topologia mais comum em redes *ad hoc*) e mesh (devido ao tipo de mobilidade inerente às redes *ad hoc*, principalmente em FANET);
- **Quantidade de VANTs:** a quantidade de VANTs foi acrescida de modo exponencial, mantendo-se apenas uma estação de controle terrestre em cada caso (16, 32, 64 e 128 VANTs);
- **Velocidade dos VANTs:** Duas diferentes velocidades foram utilizadas (baixa: 25m/s e alta: 50m/s).

Variou-se o número de VANTs com o objetivo de cobrir redes pequenas (como em ([SINGH; VERMA, 2015](#))) e também grandes. Os experimentos foram executados com 16, 32, 64 e 128 VANTs distribuídos em matrizes $n \times m$ no simulador OMNeT++ sendo 4×4 , 5×5 (mesma posição para mais do que um VANT), 8×8 , 11×11 (mesma posição para mais de um VANT) respectivamente. A distância entre cada posição vertical e horizontalmente foi fixada em

160 m. Em todos os experimentos realizados quanto maior o número de VANTs maior o impacto na degradação do desempenho da rede com exceção do IEEE 802.11n, fazendo uso do protocolo AODV, onde obteve-se altas taxas de pacotes transmitidos com sucesso.

O tempo de simulação de todos os experimentos foi de 1000 segundos que foi escolhido para prover uma simulação com o tempo similar de um tempo de voo de um VANT de pequeno porte. Os experimentos compararam as taxas de pacotes transmitidos com sucesso. Na [Figura 57](#) e [Figura 58](#) são ilustrados os resultados para o IEEE 802.11n e IEEE 802.15.4 respectivamente.

Figura 57 – Comparação dos pacotes transmitidos com sucesso pelo IEEE 802.11n.

Fonte: Adaptada de [Marconato et al. \(2016\)](#).

Figura 58 – Comparação de pacotes transmitidos com sucesso pelo IEEE 802.15.4.

Fonte: Adaptada de [Marconato et al. \(2016\)](#).

O comportamento geral de simulação IEEE 802.15.4 pode ser descrito como uma fração reduzida de pacotes transmitidos com sucesso. A aplicação do protocolo de roteamento AODV e o número crescente de VANTs na rede causou resultados ainda piores.

Por outro lado, o comportamento geral do IEEE 802.11n mudou consideravelmente com o protocolo de roteamento AODV. Sem um protocolo de roteamento a taxa de pacotes transmitidos com sucesso é semelhante aos resultados observados para IEEE 802.15.4. No entanto, com protocolo de roteamento AODV no IEEE 802.11n observa-se que quase 100% dos pacotes atingiu com sucesso o seu destino. O número de VANTs não afetou o desempenho do IEEE 802.11n com protocolo de roteamento AODV.

A Figura 59 ilustra a configuração estrela (com e sem protocolo de roteamento) e a Figura 60 a configuração *mesh* fazendo uso do protocolo AODV.

Figura 59 – Comparação de pacotes transmitidos com sucesso em configuração estrela

Fonte: Adaptada de Marconato *et al.* (2016).

Figura 60 – Comparação de pacotes transmitidos com sucesso em configuração *mesh*

Fonte: Adaptada de Marconato *et al.* (2016).

Foi também analisado o atraso fim a fim apresentado por cada um dos protocolos nas duas topologias de rede. Quanto mais VANTs foram agregados à FANETs maior foi o atraso em ambas as topologias para o protocolo IEEE 802.11n. Entretanto, na topologia estrela o parâmetro

velocidade influencia no aumento do atraso como pode ser observado nos resultados ilustrados na Figura 61.

Por outro lado, na topologia *mesh* não há diferenças nos resultados mesmo quando há modificação no parâmetro velocidade da aeronave, como pode ser observado nos resultados ilustrados na Figura 62.

Figura 61 – Análise do atraso fim a fim do IEEE 802.11n - Topologia estrela

Fonte: Adaptada de Marconato *et al.* (2016).

Figura 62 – Análise do atraso fim a fim do IEEE 802.11n - Topologia *mesh*

Fonte: Adaptada de Marconato *et al.* (2016).

A análise do atraso fim a fim para o IEEE 802.15.4 apresenta resultados diferentes. Na topologia estrela o parâmetro de incremento do atraso fim a fim é o número de aeronaves (Figura 63). Por outro lado, na topologia *mesh* os números dos atrasos são altos e não mudaram muito com o aumento do número de aeronaves como ilustrado na Figura 64.

Figura 63 – Análise do atraso fim a fim no IEEE 802.15.4 - Topologia estrela

Fonte: Adaptada de [Marconato et al. \(2016\)](#).

Figura 64 – Análise do atraso fim a fim no IEEE 802.15.4 - Topologia Mesh

Fonte: Adaptada de [Marconato et al. \(2016\)](#).

Com base em experiências reais foi possível observar que o AODV fornece um bom desempenho com o protocolo IEEE 802.11n quando se está preocupado com a quantidade de pacotes transmitidos e recebidos, considerando-se a perda de pacotes (principalmente mensagens *HELLO*). Comparando estes resultados com a simulação (mesmo cenário - 2 aeronaves e uma estação de controle terrestre) é possível averiguar o mesmo comportamento, com o protocolo AODV atingindo alta taxa de pacotes transmitidos com sucesso. Sendo assim, é possível assumir, com base nestes resultados, que o comportamento de simulação é semelhante quando o número de VANTs é aumentado. Assim, foram realizadas simulações alterando a quantidade de VANTs para avaliar o impacto das topologias em FANETs.

O problema com o protocolo IEEE 802.15.4 é visto em uma topologia *mesh* devido à tempestade de *broadcast* causada pela grande quantidade de mensagens *HELLO*. O atraso na

reconexão ocorre devido a perda de mensagens *HELLO* quando há perdas de rota ou desconexão do VANT. Isso pode ser resolvido usando o protocolo IEEE 802.11n, que oferece taxas elevadas da entrega na topologia *mesh* mesmo com um grande número de VANTs.

Outra maneira de resolver este problema é mitigar as mensagens de *HELLO*, uma vez que existe um problema de tempestade de *broadcast* ocorrendo. Essa tempestade pode ser vista como uma ameaça, uma vez que pode causar uma negação de serviço mesmo que não-intencional. Resolver este problema é possível por meio da mitigação das mensagens em ambos os protocolos (IEEE 802.15.4 e IEEE 802.11n) e topologias, uma vez que asseguram um baixo atraso fim a fim e altas taxas de entrega.

Se um dispositivo IEEE 802.11n pode se conectar diretamente à rede IP, por que pensar em usar a tecnologia IEEE 802.15.4 que requer uma ponte extra, uma vez que não é baseada em IP? As razões são de custo, consumo de energia e complexidade (FURLONG; ERICKSON, 2011). O IEEE 802.11n normalmente requer um micro-controlador ou microprocessador para evitar um gargalo de mensagens no seu tráfego e processadores melhores são mais caros. Outro problema com IEEE 802.11n é a necessidade de uma conexão constante para permitir que os dados sejam encaminhados, o que aumenta o consumo de energia que nesse tipo de veículo é preciosa.

Uma vez que uma conexão IEEE 802.11n é uma ligação sem fios constante, um software mais complexo é necessário para lidar com casos em que a ligação é interrompida. Com IEEE 802.15.4 não há nenhuma conexão que precisa ficar aberta (o dispositivo final pode simplesmente acordar, enviar a sua mensagem, aguardar uma confirmação e, em seguida, voltar a dormir), permitindo que o dispositivo possa transmitir em níveis mais elevados de energia (maior alcance) e economizar energia quando gasta menos tempo com uma conexão ativa de rádio frequência.

Uma vez considerando o uso destes protocolos e levando-se em consideração questões como consumo de energia, custo e complexidade (quando maiores os VANTs maiores as preocupações), o IEEE 802.15.4 deve ser considerada a melhor opção.

Estes resultados demonstram que, dentro de topologia em estrela mais recursos de rede são usados, portanto, ocorrem colisões. Além disso, devido à alta velocidade dos VANTs, o *link* dedicado entre cada VANT e GCS flutua e afeta as trocas de dados. Pode-se concluir que FANETs fazendo uso de topologia *mesh* com IEEE 802.11n são mais seguras do que usando topologia em estrela com o mesmo protocolo. Embora o desempenho do IEEE 802.15.4 não seja tão bom como o IEEE 802.11n na topologia *mesh*, ele deve ser considerado para uso nesta topologia devido ao baixo custo, baixo consumo de energia e alta conectividade.

A geração automática de código para o OMNeT++ a partir do uso do modelo de arquitetura LARISSA garantiu a modelagem real da aeronave. Desse modo, por meio desses experimentos foi possível demonstrar a efetividade do uso do protocolo de roteamento em diferentes tipos de protocolos de comunicação sem fio em diferentes topologias. Além disso,

foi possível validar o LARISSA uma vez que os resultados puderam ser comparados (e sendo equivalentes) aos resultados obtidos a partir da realização de voos reais.

5.3 Projeto dos experimentos da camada Controle

Outro conjunto de experimentos apresentados com objetivo de validar o LARISSA, referem-se à camada de Controle. Neste conjunto de experimentos foram realizadas as modelagens de duas placas controladoras de pilotos automáticos utilizados em VANTs reais. As placas controladoras foram modeladas e os resultados foram comparados aos modelos reais.

Por meio deste experimento é possível evidenciar algumas das vantagens oferecidas pelo LARISSA: a integridade conceitual, flexibilidade, confiabilidade, melhorando a capacidade de reutilização e manutenção, maior nível de abstração e interoperabilidade, as interfaces mais interativas entre dispositivos, e a possibilidade de gerar dados a partir do modelo que sirvam para o processo de certificação de componentes e sistemas.

A maioria dos pilotos automáticos dos VANTs disponíveis atualmente compartilha de uma arquitetura similar. Na parte de hardware utilizam um pacote clássico de sensores:

- IMU - unidade de medição de inércia é um componente eletrônico responsável por mensurar a velocidade, orientação e forças gravitacionais em uma aeronave. Para isso, é utilizada uma combinação de acelerômetros e giroscópios e em alguns casos magnetômetros;
- GPS - sistema de posicionamento global;
- Sensores de pressão, tais como barômetros e sensores de velocidade do ar;
- *Links* de rádio para comunicação com a estação de controle terrestre.

A elaboração dos cenários foi iniciada com a modelagem em UML e os artefatos que a compõe. A modelagem utilizou a metodologia 4 + 1 *views*. Diferentemente do experimento de comunicação (seção 5.2) onde somente os parâmetros são mudados, neste experimentos as controladoras são também distintas, o que faz com que as visões sejam diferentes para cada placa controladora.

5.3.1 SLUGS - Santa Cruz Low-cost UAV GNC System

O *Santa Cruz Low-cost UAV GNC System* (SLUGS)⁴ é um piloto automático para VANTs que foi desenvolvido pelo *Autonomous System Laboratory* na Universidade de Califórnia - Santa Cruz, lançado sob uma licença *open-source* do Massachusetts Institute of Technology (MIT). O objetivo do projeto SLUGS é fornecer um sistema versátil, robusto e acessível para pesquisa em pequenas aeronaves (TOEPKE, 2012; SLUGS, 2012).

⁴ <<https://slugsuav.soe.ucsc.edu>>

A arquitetura do SLUGS difere da arquitetura comum existente ao dividir as tarefas realizadas pelo piloto automático para duas unidades de processamento independentes: o *Sensor Digital Signal Controllers* (DSC) e o *Control DSC*, sendo a comunicação entre eles realizada via o protocolo SPI. Com isso, o sistema possui o dobro de capacidade de processamento se comparado com seu modelo anterior, possibilitando a implementação de algoritmos mais complexos, mas sem impactos significativos no gasto energético e em seu tamanho (LIZARRAGA, 2009; SLUGS, 2012).

A Figura 65 ilustra a arquitetura SLUGS, a qual foi modelada utilizando o *profile* LARISSA utilizando a metodologia 4+1 *views*.

Figura 65 – Arquitetura - SLUGS

Fonte: Adaptada de SLUGS (2012).

A *Use Case View*, Figura 66, ilustra os pacotes necessários para a modelagem da controladora, sendo que o pacote *ControllerBoard* representa a controladora em si, o pacote *FlightSensors* os sensores de voo e *CommunicationDevice* o pacote para definição do dispositivo utilizado no *link* de comunicação de dados.

Todos os pacotes necessários contemplados no *profile* LARISSA e aplicados na modelagem UML da controladora SLUGS são representados pela *Logical View*, ilustrada na Figura 67. A cada pacote, definidos na *Use Case View* Figura 66, é aplicado o *profile* relativo sendo, neste caso, todos pertencentes à camada Física do segmento aéreo do LARISSA.

Ao pacote *ControllerBoard* aplica-se o *profile* `larissa::UAV::Aerial::Physical::AvionicsEletronics::Controller`; ao pacote *FlightSensors* o *profile* `larissa::UAV::Aerial::Physical::AvionicsEletronics::Sensor::FlightSensor` e ao *CommunicationDevice* o *profile* `larissa::UAV::Aerial::Physical::AvionicsEletronics::Communications::AirToGround::TelecommandTelemetry`.

Figura 66 – Use Case View - SLUGS

Fonte: Elaborada pelo autor.

Figura 67 – Logical View - SLUGS

Fonte: Elaborada pelo autor.

Com a finalidade de expressar um nível maior de detalhamento a *Process View* específica como deve ocorrer a modelagem dos artefatos, ilustrando os estereótipos dos *profiles* a serem aplicados aos objetos. As mensagens *ChooseDevice* demonstram qual estereótipo do *profile* deve ser aplicado ao objeto.

As mensagens do tipo *Attach* indicam quais artefatos irão compor um objeto. Para este

cenário são apresentadas duas *Process View*: uma para os sensores, ilustrada pela Figura 68 e outra para os demais componentes (controladora e dispositivo de rede) ilustrados na Figura 69.

Figura 68 – *Process View* - *SLUGS* dos sensores

Fonte: Elaborada pelo autor.

Figura 69 – *Process View* - *SLUGS* da controladora

Fonte: Elaborada pelo autor.

A Figura 70 ilustra o pacote *FlightSensors* com os blocos SysML *IMUDevice*, *GPSDevice*, *BarometricDevice* e *MagnetometerDevice* sendo aplicados os respectivos estereótipos *IMU*, *GPS*, *barometricAltimeter* e *magneticCompass*. Também estão ilustradas as *tagged value* de cada bloco.

Similar ao pacote *FlightSensors*, a Figura 71 ilustra o pacote *CommunicationsDevice* e o bloco SysML *RadioModemDevice*. Foi aplicado o estereótipo *radioModem* e suas *tagged values* devidamente preenchidas. Com isso define-se a configuração do dispositivo onde o padrão *wireless* é o IEEE 802.15.4 (*standard=ieee802154*), a frequência de operação é 2,4 Ghz (*frequency=_2400*), a potência de transmissão é 1 mW (*rfPower=1*), a sensibilidade de recepção é -95 dBm (*sensitivity=-95*) e o canal é o 33 (*channel=33*).

Figura 70 – O Pacote *Flight Sensors - SLUGS*

Fonte: Elaborada pelo autor.

Figura 71 – O Pacote *Communication Device - SLUGS*

Fonte: Elaborada pelo autor.

O pacote *ControllerBoard*, ilustrado na [Figura 72](#), é composto pelos blocos SysML *SLUGSBoard* ao qual foi aplicado o estereótipo *board* e a *tagged value boardType* representa o tipo da placa. *AllInOne* significa que esta controladora contempla os sistemas de voo, navegação e missão. A *SLUGSBoard* é composta de dois microcontroladores representados pelos blocos *SensorDSC* e *ControlDSC* e suas *tagged values*, as quais especificam o fabricante (ma-

ker=Microship), modelo (model=DSPIC33Fj256MC710) e velocidade de *clock* (*speed*=32768). Cada microcontrolador possui sua própria memória *FLASH* sendo representada pelos blocos *MemorySensor* e *MemoryControl* e as *tagged values memoryType* e *size* especificam o tipo de memória e seu tamanho, expresso em bytes.

Figura 72 – O Pacote *Controller Board* - *SLUGS*

Fonte: Elaborada pelo autor.

A junção de todos os objetos no modelo SysML que representa graficamente a implementação do modelo final do SLUGs é ilustrada na [Figura 73](#). Esta representação remete a [Figura 65](#). O sistema como um todo é representado pelo bloco SysML SLUGs, a controladora representada pela *SLUGSBoard*, que por sua vez é composta pelo *SensorDSC* e *ControlDSC* ligados entre si pelo protocolo SPI. O sensor *IMUDevice* está ligado a outra porta SPI do *SensorDSC*, ainda neste DSC tem-se os sensores *BarometricDevice*, *MagnetometerDevice* e *GPSDevice* ligados respectivamente às portas com os protocolos ADC, I2C e RS232.

No *ControlDSC* tem-se o *NetworkDevice* conectado via RS232. Outras portas com seus respectivos protocolos estão representadas no modelo, entretanto, os dispositivos que são ali conectados não estão no escopo desta modelagem, como por exemplo as portas IC, onde se conecta o receptor do rádio controle e PWM_OC que é a saída para os atuadores de superfície móveis das aeronaves.

O modelo de implantação é abordado pela *Deployment View*, ilustrada na [Figura 74](#), e demonstra em forma de nós os módulos individuais do modelo e interdependência entre eles. No caso do SLUGs tem-se os sensores *IMU*, *GPS*, *Magenetrometer* e *Barometric*, o dispositivo de rede *NetworkDevice* e suas ligações à controladora *ControllerBoard*.

Figura 73 – Implementation View - SLUGS

Fonte: Elaborada pelo autor.

Figura 74 – Deployment View - SLUGS

Fonte: Elaborada pelo autor.

O modelo obtido foi utilizado para a confecção de uma nova placa controladora do SLUGs. Foi gerado, a partir do modelo com auxílio de uma ferramenta própria para o projeto de circuitos impressos (*Printed Circuit Board (PCB)*), o esquemático da placa (ilustrado na [Figura 75 - SensorDSC e ControlDSC](#)) e na sequência foi confeccionada a placa propriamente dita ([Figura 76](#)).

Figura 75 – SLUGS - Esquema Controladora

Fonte: Elaborada pelo autor.

Figura 76 – SLUGS - LSEC

Fonte: Elaborada pelo autor.

A nova placa foi concebida fazendo uso de sensores mais modernos do que os da placa SLUGS original. Entretanto, o funcionamento da placa foi o mesmo quando se compara os resultados obtidos com a execução do piloto automático na placa original e na placa gerada a partir do uso do LARISSA. A nova placa foi conectada e foi constatada a comunicação permitindo assim que o código do piloto automático do SLUGS fosse executado na mesma. A execução do piloto na nova placa permitiu a realização do *Hardware In The Loop Simulation* (HILS) (Figura 77), processo esse que foge do escopo desta tese e, consequentemente, a utilização

da nova placa e do piloto automático do SLUGs para a realização de voos reais.

Figura 77 – SLUGS - HILS

Fonte: Elaborada pelo autor.

Este experimento permitiu evidenciar vantagens do uso do LARISSA como: a flexibilidade e o fato do modelo ser atemporal, pois permite que novas placas sejam confeccionadas com base no mesmo modelo sendo que os avanços tecnológicos são contemplados.

5.3.2 YAPA - PAPARAZZI UAV

O objetivo principal da modelagem de uma segunda controladora foi demonstrar uma das vantagens do LARISSA, o reúso. A reusabilidade fica evidente quando observa-se que as *views* obtidas em ambas as controladoras são bastante similares.

A YAPA é um dos modelos de controladora desenvolvidos no projeto Paparazzi⁵ que contempla um sistema completo de hardware e software de código aberto para SANT. Esse sistema inclui tanto o piloto automático quanto a estação de controle terrestre, o planejamento de missão e o software de monitoramento, e um *datalink* bi-direcional para a telemetria e controle da aeronave (ENAC, 2012a). Este modelo foi escolhido dentre vários que o Paparazzi possui uma vez que o Laboratório de Sistemas Embarcados Críticos (LSEC) (Laboratório onde o projeto está inserido) possui um exemplar da segunda versão, denominada YAPA/v2.0, e com isso ser facilitado o processo de validação da especificação com o modelo real.

Como citado anteriormente, as controladoras de VANTs compartilham uma arquitetura similar. No que se refere ao hardware a maioria utiliza um pacote clássico de sensores e que

⁵ <<http://www.paparazziuav.org>>

também são abordados nesta seção.

A arquitetura da controladora YAPA utiliza um microprocessador, memória e várias portas para interligar os sensores e o dispositivo de rede(ENAC, 2012b).

A Figura 78 ilustra a arquitetura YAPA, a qual foi modelada utilizando o *profile* LARISSA e a metodologia 4+1 *views*.

Figura 78 – Arquitetura - YAPA

Fonte: Adaptada de ENAC (2012b).

A *Use Case View* da YAPA, ilustrada na Figura 79 é idêntica à *Use Case View* da SLUGS uma vez que está sendo realizado a repetição do procedimento, modelar uma controladora e seus principais sensores, assim os mesmos pacotes são necessários. A controladora, propriamente dita, necessita do pacote *ControllerBoard*, do pacote *FlightSensors* necessário aos sensores de voo e do pacote *CommunicationDevice* para definição do dispositivo de comunicação de dados.

Assim como ocorreu na modelagem da controladora do SLUGs, todos os pacotes necessários contemplados pelo *profile* LARISSA aplicados na modelagem UML da controladora YAPA são ilustrados na *Logical View* (Figura 80). O procedimento ocorre como segue: ao pacote *ControllerBoard* aplica-se o *profile* `larissa::UAV::Aerial::Physical::AvionicsEletronics::Controller`; ao pacote *FlightSensors* o *profile* `larissa::UAV::Aerial::Physical::AvionicsEletronics::Sensor::FlightSensor` e ao *CommunicationDevice* o *profile* `larissa::UAV::Aerial::Physical::AvionicsEletronics::Communications::AirToGround::TelecommandTelemetry`.

A *Process View* especifica a sequência da modelagem dos artefatos ilustrando os estereótipos dos *profiles* a serem aplicados aos objetos. As mensagens *ChooseDevice* demonstram qual estereótipo do *profile* deve ser aplicado ao objeto do tipo sensores.

As mensagens *ChooseBord* e *ChooseMicrocontroller* especificam os estereótipos Flight-

Figura 79 – Use Case View - YAPA

Fonte: Elaborada pelo autor.

Figura 80 – Logical View - YAPA

Fonte: Elaborada pelo autor.

Board e MicroController, respectivamente. As mensagens do tipo *Choose_Memory* representam estereótipos que definem os tipos de memória utilizados na controladora. As mensagens do tipo *Attach* indicam quais artefatos irão compor um objeto. Para este cenário são apresentados três *Process View*, duas para os sensores (Figura 81 e Figura 82) e outra para os demais componentes, a controladora e dispositivo de rede (Figura 83).

Figura 81 – *Process View* - YAPA dos sensores - parte 1

Fonte: Elaborada pelo autor.

Figura 82 – *Process View* - YAPA dos sensores - parte 2

Fonte: Elaborada pelo autor.

A [Figura 84](#) ilustra o pacote *FlightSensors* com os blocos SysML *GyroscopeDevice*, *GPSDevice*, *BarometricDevice*, *AccelerometerDevice*, *MagnetometerDevice* e *AirSpeedDevice*. Nesses blocos foram aplicados os estereótipos *gyroscope*, *GPS*, *barometricAltimeter*, *accelerometer*, *magneticCompass* e *aerodynamicSpeed*, respectivamente. Também estão ilustrados as *tagged value* de cada bloco.

O pacote *CommunicationsDevice*, ilustrado na [Figura 85](#), exibe o bloco SysML *RadioModemDevice* sendo o *radioModem* o estereótipo aplicado. Suas *tagged values* estão devidamente preenchidas, definindo a configuração do dispositivo onde o padrão *wireless* é o ZigBee (*standard*=ZigBee), a frequência de operação é 2,4 Ghz (*frequency*=_2400) potência de transmissão é 1 mW (*rfPower*=1), a sensibilidade de recepção é -95 dBm (*sensitivity*=-95) e o canal é o 27 (*channel*=27).

A [Figura 86](#) ilustra o pacote *ControllerBoard* que é composto pelos blocos SysML *YAPA2Board* ao qual foi aplicado o estereótipo *board*. A *tagged value* *boardType* representa o tipo da controladora e o *AllInOne* os sistemas de voo, navegação e missão. A *YAPA2Board* é composta de um microcontrolador representado pelo bloco *processor* e suas *tagged values* que especificam:

Figura 83 – *Process View - YAPA* da controladora

Fonte: Elaborada pelo autor.

Figura 84 – O Pacote *Flight Sensors* - YAPA

Fonte: Elaborada pelo autor.

fabricante (*maker*=ARM), modelo (*model*=LPC2148) e velocidade de *clock* (*speed*=60000). Tem-se a memória principal *MainMemory* do tipo RAM (*memoryType*=RAM) e capacidade de 32

Figura 85 – O Pacote *Communication Device* - YAPA

Fonte: Elaborada pelo autor.

KB (*size=32000*), a memória secundária (*SecondaryMemory*) do tipo RAM (*memoryType=RAM*) com capacidade de 8 KB (*size=8000*) e uma memória auxiliar (*memoryType=FLASH*) com capacidade de 512 KB (*size=512000*).

Figura 86 – O Pacote *Controller Board* - YAPA

Fonte: Elaborada pelo autor.

A *Implementation View* representa o modelo final que é ilustrado na [Figura 87](#). Esta representação remete à [Figura 78](#). A controladora ligada aos sensores e dispositivos de rede é representada pelo bloco SysML YAPA, e o componente *YAPA2Board* representa a controladora.

Os sensores *GyroscopeDevice*, *BarometricDevice*, *AccelerometerDevice*, *MagnetometerDevice* e *AirSpeedDevice* estão ligados à controladora utilizando portas GPIO. O dispositivo de rede *NetworkDevice* e o sensor *GPSDevice* são conectados às portas seriais. Outras portas com seus respectivos protocolos estão representadas no modelo, entretanto, os dispositivos que são ali conectados não estão no escopo desta modelagem como, por exemplo, as portas PPM, onde se conecta o receptor do rádio controle e PWM de 0 à 7 que são as saídas geralmente utilizadas para conectar os atuadores de superfície móveis das aeronaves.

Figura 87 – Implementation View - YAPA

Fonte: Elaborada pelo autor.

O modelo de implantação é abordado pela *Deployment View* (ilustrada na Figura 88) e demonstra em forma de nós os módulos individuais do modelo e a interdependência entre eles. No caso da YAPA tem-se os sensores *AirSpeed*, *GPS*, *Magenetrometer*, *Barometric*, *Gyroscope* e *Accelerometer*, além do dispositivo de rede *NetworkDevice* e suas ligações à controladora *ControllerBoard*.

A geração da controladora YAPA (Figura 89) seguiu os mesmos passos da controladora do SLUGs. O esquemático do processador é ilustrado na Figura 90 e é possível observar a similaridade do mesmo com o modelo gerado na *Implementation View*. Testes de HILS foram executadas com a YAPA (também fora do escopo deste trabalho), mas permitiram comprovar efetividade do modelo para a geração da controladora. Novos experimentos fazendo uso não só da controladora YAPA, mas de diversos outros componentes do PAPARAZZI, estão sendo

Figura 88 – Deployment View - YAPA

Fonte: Elaborada pelo autor.

executados e planejados em colaboração com o grupo francês que concebeu o PAPARAZZI, vide [seção 6.4](#).

Figura 89 – YAPA - LSEC

Fonte: Elaborada pelo autor.

5.4 Projeto dos experimentos da Camada Física do Segmento Aéreo

O projeto *Autonomous Vertical takeOff and laNding* (AVALON) ([SILVA; MARCONATO; BRANCO, 2015](#)) trata da concepção de um VANT de asas fixas capaz de decolar e pousar verticalmente sem a necessidade de uma pista. Consiste em uma pequena aeronave dotada

Figura 90 – YAPA - Esquema

Fonte: Elaborada pelo autor.

de motor elétrico, que quando em solo repousa por um sistema de suporte montado na cauda da aeronave, permitindo sua decolagem e pouso vertical. Na [Figura 91](#) é ilustrado seu *design* em diferentes perspectivas.

Como o modelo de arquitetura LARISSA contempla a camada física da aeronave, sendo possível especificar dentre outras as classes aeronave, o conjunto propulsor e combustível utilizado e ainda controladora de voo como apresentado nas [subseção 5.3.1](#) e [subseção 5.3.2](#), demonstra-se nesta subseção a modelagem destes componentes aplicados ao AVALON, com o objetivo de comprovar que o LARISSA abrange todos os componentes que compõem o SANT.

O desenvolvedor do AVALON deve definir a aeronave, o sistema de propulsão e a controladora de voo. Esta definição dá-se por meio da criação de pacotes a saber: *Aircraft*, *PropulsionSystem* e *ControllerBoard*, ilustrados pela *Use Case View*, [Figura 92](#).

Figura 91 – AVALON - Design

Fonte: Adaptada de [Silva, Marconato e Branco \(2015\)](#).

Figura 92 – Use Case View - AVALON

Fonte: Elaborada pelo autor.

Na [Figura 93](#) é representada a *Logical View* do AVALON ilustrando os *profiles* a serem aplicados: para o pacote *Aircraft*, *larissa::UAS::Aerial::Physical::Structure* e *larissa::UAS::Aerial::Physical::AuxiliarySystems*, para o pacote *PropulsionSystem*, o *profile* *larissa::UAS::Aerial::Physical::Energy::Propulsion* e *larissa::UAS::Aerial::Physical::Energy::Fuel* e ao pacote

ControllerBoard aplica-se o *profile larissa::UAV::Aerial::Physical::AvionicsEletronics:: Controller*.

Figura 93 – Logical View - AVALON

Fonte: Elaborada pelo autor.

A sequência da modelagem dos artefatos ilustrando os estereótipos dos *profiles* a serem aplicados aos objetos é representada pela *Process View*. As mensagens *ChoosePropeller*, *ChooseBattery*, *ChooseBord*, *ChooseMicrocontroller*, *ChooseMemory*, *ChooseAircraft* especificam os estereótipos *Propeller*, *Battery*, *FlightBoard*, *Memory*, *MicroController* e *Aircraft*, respectivamente. As mensagens do tipo *Attach* indicam quais artefatos irão compor um objeto. Para este cenário são apresentados três *Process View*, uma para o sistema de propulsão (Figura 94, uma para a controladora Figura 95) e outra para a aeronave (Figura 96).

Os blocos SysML *ElectricEngine* e *Batteries* compõem o pacote *PropulsionSystem* especificando o conjunto moto propulsor do AVALON. O estereótipo *propeller* indica que o motor da aeronave é da classe que utiliza hélices para gerar empuxo e a *tagged value type* especifica o tipo de motor, neste caso *ElectricMotor*. Os motores elétricos de pequenas aeronaves, como o AVALON, utilizam como combustível as baterias representadas no pacote pelo bloco

Figura 94 – Process View - AVALON do sistema de propulsão

Fonte: Elaborada pelo autor.

Figura 95 – Process View - AVALON da controladora

Fonte: Elaborada pelo autor.

Batteries. A este bloco é aplicado o estereótipo *battery*, sendo que as *tagged values* definem as características física das baterias. Voltagem: 14,8V (*voltage*=14.8), amperes: 5.000mA (ampere=5000), número de células que compõe a bateria: 4, (*numCell*=4) e a composição química: Li-Po (*cellChemetry*=LiPo). Na [Figura 97](#) é ilustrado o pacote *PropulsionSystem*.

O pacote *ControllerBoard*, ilustrado na [Figura 98](#), é composto pelos blocos SysML *OdroidXU4* ao qual foi aplicado o estereótipo *board*. A *tagged value* *boardType* representa o tipo da controladora e o *Flight* o sistema de voo. A *OdroidXU4* é composta de um microcontrolador representado pelo bloco *processor* e suas *tagged values* que especificam: fabricante

Figura 96 – Process View - AVALON da aeronave

Fonte: Elaborada pelo autor.

Figura 97 – O Pacote *PropulsionSystem* - AVALON

Fonte: Elaborada pelo autor.

(*maker*=SamSung), modelo (*model*=Exynos5422) e velocidade de *clock* (*speed*=2000000). Tem-se a memória principal *MainMemory* do tipo RAM (*memoryType*=RAM) e capacidade de 2 GB (*size*=2000000000).

A Figura 99 ilustra o pacote *Aircraft*, onde o bloco *Airplane* utiliza o estereótipo *aircraft* para classificar o tipo de aeronave e o estereótipo *takeoffGear* para representar o sistema auxiliar de decolagem. O AVALON pertence a classe aeronaves de Asa Fixa (*classification*=*FixedWing*) e o sistema de decolagem apoiado na cauda (*takeoffType*=*Tailsitter*).

Figura 98 – O Pacote *Controller Board* - AVALON

Fonte: Elaborada pelo autor.

Figura 99 – O Pacote *Aircraft* - AVALON

Fonte: Elaborada pelo autor.

O modelo SysML final do AVALON, ilustrado [Figura 100](#), é contemplado pela *Implementation View*. À aeronave (*Airplane*) estão conectadas a controladora (*OdroidXU4*) e o motor (*ElectriEngine*) que faz uso das baterias (*Batteries*).

Por fim, na [Figura 101](#) é ilustrada a *Deployment View*, visão que representa em forma de nós o produto da modelagem individualmente e a interdependência entre os mesmos. O nó *Aircraft* ilustra a estrutura da aeronave sem componentes. Os componentes que devem compor a

Figura 100 – Implementation View - AVALON

Fonte: Elaborada pelo autor.

aeronaves são: (*Engine*), que necessita de combustível para funcionar (*Energy*) e a Controladora (*ControllerBoard*).

Uma aeronave não é constituída somente dos componentes exibidos nesta subseção, entretanto, para o escopo deste caso de uso demonstrou-se os componentes modelados e testados no simulador X-Plane. Uma vez que o LARISSA é totalmente flexível, pode-se modelar somente as partes de interesse e conforme o projeto do AVALON avança e surge a necessidade de atingir um nível de maior profundidade na camada física ou em qualquer outra camada, basta editar o modelo SysML do AVALON e aplicar o *profile* LARISSA nos objetos pertinentes.

Detalhes da simulação do AVALON podem ser encontrados em (SILVA; MARCONATO; BRANCO, 2015) uma vez que fogem ao escopo deste trabalho.

Com todos os experimentos realizados é possível afirmar que o modelo de arquitetura LARISSA auxilia no processo de desenvolvimento de SANTs. Além disso, o desenvolvimento e a utilização do *larissamodel2simulation* permitiu a concepção de um simulador de redes FANETs fazendo uso da geração automática de código para o OMNeT++. O simulador desenvolvido e denominado AVENS é detalhado na próxima seção.

5.5 AVENS - Aerial VEHICLE Network Simulator

Com base em avanços recentes, pesquisadores e desenvolvedores têm utilizado simuladores que oferecem não apenas simulação de protocolo de rede, mas também padrões realistas

Figura 101 – *Deployment View - AVALON*

Fonte: Elaborada pelo autor.

de modelos de mobilidade para veículos reais.

Em termos de simulação, a maioria dos simuladores de rede estão focados em protocolos tradicionais de *Mobile Ad Hoc Networks* (MANETs) e alguns deles estendem os protocolos para *Vehicular Ad Hoc Networks* (VANETs). Apesar de haver outros simuladores para redes de veículos terrestres, o *Vehicular in Network Simulation* (VEINS)⁶ é o mais utilizado e aceito em pesquisas recentes (SOMMER, 2014).

Por meio de análise da literatura aberta não há indícios de que exista um simulador capaz de representar modelos de mobilidade realistas para redes de VANTs, uma vez que os protocolos atuais e modelos de mobilidade assumem características de antenas isotrópicas ou omnidirecional, sendo assim as características específicas e relativas ao cenário aéreo não são plenamente atendidas (BOUKERCHE *et al.*, 2008). Além disso, esses simuladores não estão suficientemente equipados para simular adequadamente a mobilidade de aeronaves em movimento, nem a altura e orientação imposta por uma rede aérea.

Pesquisas sobre redes de VANTs demonstram apenas os resultados com base em simulações gerais da rede. A maioria das pesquisas recentes apontam que é obrigatório que VANTs

⁶ <<http://veins.car2x.org/>>

cooperem entre si a fim de realizar missões complexas (REN; BEARD; ATKINS, 2007; WANG; XIN, 2013). Portanto, é necessária a implementação de uma rede que possa lidar com esses novos padrões conceituais de rede, ou seja, que possam lidar com as FANETs.

O OMNeT++ é um simulador que fornece infraestrutura e ferramentas para a composição de um cenário de simulação (como já pode ser observado por meio dos experimentos já apresentados na seção 5.2), apesar dos modelos de mobilidade existentes não representarem completamente os modelos das FANETs, este simulador permite sua implementação e integração com outras ferramentas (YANMAZ; KUSCHNIG; BETTSTETTER, 2013; YANMAZ *et al.*, 2014; BOUACHIR *et al.*, 2014).

Considerando a falta de simuladores para FANETs, a necessidade de explorar características dos VANTs mais amplamente e a facilidade provida pelo *lasissamodel2simulation*, o simulador AVENS foi concebido. O propósito do AVENS é prover uma plataforma de análise de redes móveis *ad hoc* em que os VANTs são os nós móveis que compartilham o meio para troca de mensagens. O objetivo é utilizar um simulador de voo que controle os veículos aéreos e um simulador de rede, permitindo obter métricas sobre o funcionamento da rede, tais como, taxa transmissão, perda de pacotes, *throughput*, entre outras, além de permitir obter informações relativas à própria aeronave (por meio do uso de simuladores de voo).

Foram selecionadas duas plataformas para compor o AVENS, o OMNeT++ como o componente de rede o *X-Plane Flight Simulator*⁷.

O X-Plane foi escolhido pois é considerado o simulador de voo com a maior fidelidade física e funcional entre os motores de simulação (CRAIGHEAD *et al.*, 2007). Além disso, o X-Plane permite o uso de *plugins* desenvolvidos em C++ (X-PLANE, 2014) necessários à integração entre os simuladores. Ademais, o X-Plane permite simular condições meteorológicas (vento, chuva, turbulência) e falhas em componentes das aeronaves, os quais influenciam na trajetória de voo perfazendo uma simulação mais realistas.

Uma vez que a trajetória executada no X-Plane é enviada ao OMNeT++, este formato permite a modelagem e desenvolvimento realísticos de protocolos e modelos de mobilidade para FANETs. A integração entre os simuladores se dá por um *plugin* desenvolvido para o X-Plane e um módulo desenvolvido para o OMNeT++, ambos responsáveis pela troca de informação, que atualmente é realizada via um arquivo *eXtensible Markup Language* (XML).

O funcionamento básico do AVENS (Figura 102) segue as seguintes etapas:

1. Os parâmetros de rede dos VANTs são definidos utilizando o LARISSA;
2. Os parâmetros de rede são gerados automaticamente para o OMNeT++;
3. Os parâmetros iniciais, como número de VANTs e suas posições iniciais são informados ao X-Plane;

⁷ <<http://www.x-plane.com>>

4. É executada a simulação do voo, podendo ocorrer interferências tais como vento e falhas de funcionamento em uma ou mais aeronaves;
5. São atualizadas informações de posicionamento e velocidade da aeronaves no OMNeT++;
6. Os eventos da simulação de rede são armazenados para futura análise;
7. O processo fica em repetição (*loop*) até o fim da simulação;
8. Outras informações relativas ao voo, configuração das aeronaves, falhas programadas e condições climáticas, podem ser alteradas durante a simulação.

Figura 102 – Estrutura do AVENS - integração entre os simuladores

Fonte: Elaborada pelo autor.

Neste estudo de caso, o piloto automático e uma interface de rede sem fio foram modelados para executar simulações de comunicação de rede no OMNeT++. Os parâmetros de simulação foram gerados a partir do modelo de arquitetura LARISSA usando o gerador de código do *larrissamodel2simulation*.

5.5.1 Modelagem utilizando o profile LARISSA

A modelagem SysML do VANT foi obtida do LARISSA. Uma placa de piloto automático foi modelada, composta por um processador, uma memória principal, uma memória auxiliar, uma entrada/saída de propósito geral (interface *General Purpose Input/Output* (GPIO)), e um dispositivo de rede sem fios (já apresentada na [seção 5.3](#)). Como a comunicação de rede é o foco

deste estudo de caso outros componentes do piloto automático não são levados em consideração. Todos os parâmetros foram modelados com base em um piloto automático verdadeiro, o Paparazzi, um sistema de software e hardware abertos (ENAC, 2012a).

Além do pacote principal, dois pacotes de placas *ControllerBoard* foram criados. Esses pacotes contêm as classes que definem o piloto automático *larissa::UAS::Aerial::Physical::AvionicsElectronics::Controller* no qual o *profile* é aplicado.

Para os pacotes *AutoPilot*, *Processor*, *MainMemory* e *AuxiliaryMemory*, os seguintes estereótipos são respectivamente aplicados: *board*, *microcontroller*, *memory* e *memory* novamente. O diagrama em SysML que representa esse modelo é ilustrado na Figura 103.

Figura 103 – Pacote *ControllerBoard* com o *profile Controller*.

Fonte: Elaborada pelo autor.

O mesmo método foi aplicado ao pacote *CommunicationDevice* usando o *profile larissa::UAS::Aerial::Physical::AvionicsElectronics::Communications::AirToGround::Telecommand Telemetry*.

No bloco *RadioModem*, o estereótipo *radioModem* foi aplicado. Na Figura 104 é ilustrado o bloco e algumas *tagged values* tais como *frequency*, *rfPower* e *sensitivity*, que são utilizadas como parâmetros para representar as frequências de transmissão (433 MHz, 900 MHz, etc.), a capacidade de transmissão (1 mW, 10 mW, 50 mW, etc.) e o limiar do sinal de atenuação em dBm (-95 dBm). Todos foram utilizados para configurar a camada física do OMNeT++.

O modelo é finalizado fazendo uso do diagrama de blocos interno do SysML ilustrado na Figura 105, na qual é adicionado um bloco *AutoPilot* e um bloco *RadioModem* conectando-os por uma porta GPIO. GPIO foi modelado ao invés de uma porta serial porque: (i) o piloto automático

Figura 104 – Dispositivo de *RadioModem*

Fonte: Elaborada pelo autor.

Paparazzi tem uma porta dedicada para o *link* do piloto automático e o *radio modem* e; (ii) O modelo LARISSA representa não somente os dispositivos, mas os *links* entre eles com o objetivo de padronizar o desenvolvimentos dos SANTs.

Figura 105 – Diagrama de blocos internos - VANTs

Fonte: Elaborada pelo autor.

Existem várias maneiras de construir este tipo de modelo, a partir de uma representação gráfica simples de componentes UML. No caso específico deste trabalho foi utilizado o *larissamodel2simulation* que gera códigos automáticos a partir do modelo para o OMNeT++.

Especificamente para o OMNeT++ desenvolveu-se um módulo denominado *fileHandler* que deve ser inserido nas simulações que realizam a integração com o X-Plane. Esse módulo é o responsável por determinar o nome do arquivo XML e as variáveis que indicam o nome da variável *host*, que por sua vez identifica o VANT e número de VANTs utilizados na simulação. Em relação à mobilidade dos nós, no caso dos VANTs, houve a necessidade de estender o módulo *MovingMobilityBase* do *framework* INET, originando um novo módulo, o *Arbitrary Mobility*. Neste novo módulo foram acrescentados três novos atributos, necessário para uso no arquivo

NED, que define o *host*, *dXPosition*, *dYPosition* e *dZPosition*, que são utilizados para atualizar a posição do nó, invocando o método sobrecarregado *move()*. Na Figura 106 é exibido o diagrama de classes do módulo *Arbitrary Mobility*.

Figura 106 – Diagrama de classes *Arbitrary Mobility*

Fonte: Elaborada pelo autor.

O módulo *fileHandler* é o responsável pela criação do arquivo XML, que contém o número de VANTs que serão simulados no X-Plane e implementa uma máquina de estados simples que (i) aguarda o X-Plane reconhecer o arquivo XML, (ii) lê as coordenadas de referência do arquivo XML e; (iii) altera as posições dos VANTs no OMNeT++, repetindo o processo até a simulação terminar.

Para o X-Plane, um *plugin* foi desenvolvido com a finalidade de gravar no arquivo XML a posição de todos os VANTs. Na Figura 107 é ilustrado o diagrama de classes do *plugin*. Todos os VANTs são modelados pela classe *Aircraft*, a qual possui atributos que armazenam todos os dados relativos à posição. Esta classe é especializada em duas outras: *User Aircraft*, que modela a aeronave controlada pelo usuário no X-Plane e *MultiAircraft*, que modela outras aeronaves no simulador.

O fluxo de comunicação entre o OMNeT++ e X-Plane para o funcionamento do AVENS é ilustrado na Figura 108. Primeiro o método *initialize()* do módulo *fileHandler* do OMNeT++ cria o arquivo XML, informando qual o número de aeronaves será simulada. Após a inicialização, o módulo envia repetidamente a mensagem *WAIT_XPLANE* até que o X-Plane seja inicializado, reconheça o arquivo e leia as variáveis de configuração que constituem os passos 1 e 2 da Figura 108.

O X-Plane, por sua vez, grava no XML as posições das aeronaves sendo que no início tem-se a mensagem *READ_REFERENCES* seguida das posições e, por fim, a mensagem *UPDATE_POSITIONS* que solicita atualização das posições das aeronaves no OMNeT++. Este processo se repete em forma de *loop*, até o fim da simulação, passos 3 e 4 da Figura 108.

Figura 107 – Modelagem da aeronave no *plugin* do X-Plane

Fonte: Elaborada pelo autor.

Figura 108 – Fluxo de comunicação entre OMNeT++ e X-Plane.

Fonte: Elaborada pelo autor.

5.5.2 Estudo de Caso

A disponibilidade do *profile* LARISSA para geração automática de código para o OMNeT++ associado ao *plugin* para o simulador X-Plane permitiu a integração de todos os recursos necessários para a concepção do AVENS.

Oito cenários de simulação, cujos parâmetros são detalhados na [Tabela 8](#), foram definidos para o AVENS com intuito de validá-lo e, conseqüentemente, validar o LARISSA. Intervalo médio de envio, potência de transmissão, sensibilidade do receptor, taxa de transferência, tamanho

do campo, número de aeronaves e velocidade da aeronave são os parâmetros variados durante as simulações no AVENS.

Tabela 8 – Valores dos parâmetros para os cenários de simulação implementados no AVENS

Parâmetros Fixos	Valores							
Frequência	2.4 GHz							
Tipo de Mobilidade	Linear Mobility							
Algoritmo de Roteamento	Wise Route							
Padrão da Camada Física	802.15.4							
Tamanho <i>Playground</i> (x)	1,6 km							
Tamanho <i>Playground</i> (y)	1,6 km							
Tamanho <i>Playground</i> (z)	1,6 km							
Tempo de Simulação	3 minutos							
Número de GCS	1							
Taxa de Transferência	250 Kbps							
Pacotes enviados	170							
Intervalo médio de envio	1 segundo							
	Simulações Realizadas							
Parâmetros Variáveis	1	2	3	4	5	6	7	8
Número de UAVs	3				5			
Potência de Transmissão (mW)	100		200		100		200	
Sensibilidade do Receptor (dBm)	-104	-150	-104	-150	-104	-150	-104	-150

Fonte: Adaptada de Marconato *et al.* (2017).

Sabe-se que o *larissamodel2simulation* faz uso da estrutura MiXiM (MIXIM, 2014), que fornece suporte para dispositivos sem fio, tais como aqueles que implementam o padrão IEEE 802.15.4 utilizado neste estudo de caso. Na geração do ambiente de simulação um dos passos é criar o arquivo *omnetpp.ini* que contém uma série de parâmetros e instruções que orientam a simulação.

O procedimento é para inserir uma linha neste arquivo com a expressão "*include uav-conf.ini*" refere-se ao arquivo de configuração gerado pelo *larissamodel2simulation*. Portanto, quando a simulação é executada os parâmetros que foram especificados na modelagem e foram submetidos à transformação *larissamodel2simulation* são lidos e utilizados a partir do *uavconf.ini*.

O *larissamodel2simulation* obtém características específicas da aeronave a partir do modelo SysML que usa o *profile* LARISSA. Tais informações são utilizadas para geração de código e possibilitam que o modelo se torne confiável com base nos dados obtidos diretamente do simulador. O modelo pode ser melhorado por meio da repetição do ciclo, obtendo, por conseguinte, um modelo ideal.

O *plug-in* dentro do X-Plane é responsável pela conversão de arquivos do OMNeT++ em um conjunto de especificações detalhadas que serão utilizadas pelo simulador de voo. O *plug-in*

também garante atualizações de parâmetros tanto no OMNeT++ quanto no X-Plane durante toda a simulação.

A trajetória do VANT pode ser simulada com o AVENS. A latitude, longitude e altitude foram recuperados do X-Plane e convertidas para o quadro de navegação representada como *North-East-Up* fazendo uso do MATLAB[®]. Devido à flexibilidade e robustez do AVENS é possível visualizar a trajetória de voo de um único VANT (o caso é ilustrado na [Figura 109](#)) ou até mesmo de vários VANTs compartilhando o mesmo espaço.

Figura 109 – Representação da trajetória de navegação - *North-East-Up*

Fonte: Adaptada de [Marconato et al. \(2017\)](#).

Os gráficos, ilustrados nas [Figura 110](#) e [Figura 111](#), apresentam oito cenários de simulação. Os parâmetros de cada uma das simulações são listados na [Tabela 8](#).

Essas simulações apresentam VANTs voando em cooperação, então, apenas o *eixo - y* é modificado pela simulação. Entretanto, o AVENS não é limitado por esta configuração e deve lidar com diferentes caminhos em todas as direções incluindo os eixos *x* e *z*, se necessário. Cada simulação foi executada por 3 minutos, que é o tempo necessário para que o VANT cubra 1,6 km com uma velocidade aproximada de 8,8 m/s. O espaço para o experimento é o mesmo em todas as simulações (1,6 km x 1,6 km x 1,6 km) bem como os pacotes transmitidos por nós (170 pacotes). A potência de transmissão varia entre 100 e 200 mW, e a sensibilidade de recebimento varia entre -104 e -150 dBm. Tais valores foram escolhidos seguindo as configurações específicas do IEEE 802.15.4, permitindo a obtenção de um ambiente ideal.

Na [Figura 110](#) são ilustradas as simulações de 1 - 4 com 3 VANTS e 1 Estação de Controle Terrestre. Devido à distância entre os VANTs que cooperam para cobrir toda a área, a capacidade de transmissão e o parâmetro de sensibilidade apresentam um papel importante nos resultados. Entretanto, é possível apontar diferenças significativas nos resultados das simulações

Figura 110 – Resultados da simulação dos cenários com 3 VANTs voando em cooperação

Fonte: Adaptada de [Marconato et al. \(2017\)](#).

Figura 111 – Resultados das simulações dos cenários com 5 VANTs voando em cooperação

Fonte: Adaptada de [Marconato et al. \(2017\)](#).

5 a 8 com 5 VANTs cooperando para cobrir a mesma área, como ilustrado na [Figura 111](#). Tal comportamento é esperado uma vez que a distância entre os VANTs é menor se comparada com as simulações de 1 a 4.

Na [Figura 110](#) e [Figura 111](#) são apresentados resultados que corroboram com as contribuições do AVENS para pesquisadores e desenvolvedores de sistemas robóticos aéreos, uma vez que auxiliam na análise prévia do comportamento das redes de VANTs (FANETs) e, adem especificamente quanto a rede é afetada quando da escolha de dispositivos de hardware adequados.

É importante salientar que o objetivo principal dos experimentos foi o de avaliar o AVENS como um simulador para propósitos robóticos, não o desempenho da rede ou seu comportamento. Entretanto, é sabido que nenhum requisito pode ser avaliado isoladamente dos demais. Esta é a razão pela qual os parâmetros de rede foram explorados, apesar de não serem o foco principal. O comportamento da rede pode ser explorado em novas simulações que foquem na melhoria da rede propriamente dita.

AVENS completa a lacuna apresentada por ([BEKMEZCI; SAHINGOZ; TEMEL, 2013](#)) provendo uma ferramenta de simulação para múltiplos VANTs e protocolos de redes de propósito mais gerais e que permitam a simulação de FANETs. O AVENS pode modelar estruturas de mobilidade de múltiplos VANTs bem como diferentes protocolos de redes, sensores e outros. As informações mais atuais sobre o AVENS estão disponíveis no *site* da documentação oficial, <http://www.lsec.icmc.usp.br/>.

5.6 Considerações Finais

Neste capítulo foram descritos os experimentos realizados com objetivo de avaliar o LARISSA, bem como foi apresentado o AVENS. A execução das simulações e verificação dos resultados demonstram que tanto o LARISSA quanto o *larissamodel2simulation* e o AVENS são ferramentas úteis pois, sem o uso destas a forma de se averiguar e analisar as configurações e indicar as ideais seria por meio da realização de voos reais, o que implicaria em custos elevados e maior tempo para a execução dos testes e obtenção dos resultados.

Foram apresentados experimentos em quatro cenários distintos para validação da sub-camada de Comunicação, dois experimentos para validação da camada de Controle e um experimentos para validação da camada física do segmento aéreo.

O simulador de FANETs, denominado AVENS, foi apresentado e experimentos foram realizados para validar seu uso. A concepção do AVENS foi possível graças ao *larissamodel2simulation* e ao *profile* LARISSA, que atestam sua importância.

Os resultados obtidos validam o LARISSA, demonstrando que este reúne os requisitos para modelar o domínio de um SANT. Além disso demonstram a eficiência do LARISSA,

corroborando a hipótese que o uso de um modelo de arquitetura permite a confecção de VANTs auxiliando a sua inserção no espaço aéreo uma vez que facilitam o processo de certificação.

Sabe-se que o processo de certificação é algo complexo, e somente um modelo de arquitetura está longe de prover todos os requisitos e elementos necessários para a mesma. Entretanto, como anteriormente mencionado, segundo a [ANAC \(2010\)](#) o processo de certificação de uma aeronave passa por diversas fases e em uma delas, a Fase de Certificação do Projeto, o uso de simulações em computadores é aceito para compor o documento de avaliação da engenharia. Sendo assim, este trabalho auxilia o processo de certificação provendo dados para pelo menos uma das fases definidas pela ANAC, pois uma vez que os valores das configurações foram provenientes de *profile* LARISSA, o desenvolvedor é guiado na escolha dos componentes do domínio não permitindo que deixem de configurar atributos essenciais para a modelagem do SANT.

No próximo capítulo são apresentadas as principais contribuições desta tese, as conclusões, as limitações da metodologia, as dificuldades encontradas, a produção científica resultante e os trabalhos futuros.

CONCLUSÃO

O uso de VANTs no espaço aéreo segregado e não segregado vem aumentando, levando a agências reguladoras de diversos países, tais como FAA dos EUA e ANAC do Brasil, a buscar novas normas que regularizem seu uso civil. A ANAC, até o momento da elaboração desta tese, não publicou oficialmente a resolução que trata do assunto, entretanto, em uma prévia da legislação que foi à consulta pública no final do ano de 2015 consta que todo RPA acima de 150Kg deve passar periodicamente, pelos procedimentos de certificação, assim como acontece com as aeronaves tripuladas. Os RPAs que pesam entre 25 e 150 Kg devem passar pelo processo de certificação ao menos uma vez.

O processo de certificação de uma aeronave não é trivial, sendo composto de várias etapas (ANAC, 2010). Uma delas é a Certificação de Projetos que permite a obtenção de dados a partir de simulação de computadores. Neste sentido, esta tese apresentou e validou o modelo de arquitetura LARISSA que visa auxiliar no processo de certificação de VANTs e seus componentes. O *profile* LARISSA foi especificado a partir do modelo LARISSA e deu origem a ferramenta *larissamodel2simulation*, que em conjunto com o simulador OMNeT++, mostrou-se viável para pesquisas relacionadas à comunicação em FANETs. Experimentos realizados com diferentes cenários de simulação permitiram não só validar o LARISSA como também obter resultados relevantes quanto ao estudo de FANETs.

O simulador AVENS foi concebido a partir dos resultados com as simulações de FANETs e da necessidade de simuladores que contemplem não somente características de rede mas que levem em consideração as características inerentes da aplicação e dos nós móveis (neste caso os VANTs). O AVENS une o *larissamodel2simulation* e os simuladores OMNeT++ e X-Plane, permitindo assim que os resultados obtidos com a simulação sejam mais próximos dos resultados obtidos com voos reais de VANTs.

O modelo de arquitetura LARRISA permite que pesquisadores possam desenvolver os componentes de um SANT de forma isolada com diferentes configurações, além de utilizar

componentes já desenvolvidos e certificados com a garantia que serão compatíveis entre si.

A validação completa do modelo de arquiteturas foi realizada por meio da colaboração no desenvolvimento de um novo VANT, denominado AVALON, com base no LARISSA. Além disso, também foram realizadas comparações do modelo resultante de SANTs reais, como o PAPARAZZI.

6.1 Principais Resultados e Contribuições

A principal contribuição deste doutorado é a definição do modelo de arquitetura LARISSA. Entretanto outros foram os resultados e contribuições obtidos com o desenvolvimento desta tese de doutorado:

- **profile LARISSA:** o *profile* LARISSA é a materialização do modelo de arquitetura em uma linguagem padrão que permite a construção dos projetos de SANT propriamente ditos (MARCONATO *et al.*, 2016).
- **Simulador AVENS:** um simulador de redes de VANTs com a geração automática de código produzido a partir do desenvolvimento do modelo de arquitetura LARISSA (MARCONATO *et al.*, 2017).
- **Uso de simulação para certificação de VANTs:** demonstrou-se, assim como apregoa a ANAC (ANAC, 2010) o uso de simulação para validar o modelo de arquitetura e consequentemente prover dados para fases de certificação (SILVA; MARCONATO; BRANCO, 2015).
- **larissamodel2simulation:** desenvolvimento do *larissamodel2simulation* que permite a geração automática de código para o OMNeT++ garantindo a fidelidade ao modelo e evitando falhas na transcrição entre modelo e simulador (MARCONATO *et al.*, 2016).
- **LARISSA:** especificação e desenvolvimento de um modelo de arquitetura em camadas. Esse modelo permite que pesquisadores, desenvolvedores e fabricantes façam uso da arquitetura em seus projetos, utilizando a SysML. A divisão do modelo de arquitetura em camadas traz benefícios no que se refere a restrição das funções a cada camada, levando a padronização, facilitando a manutenção e o reúso de artefatos que compõem o SANT (MARCONATO *et al.*, 2014; MARCONATO *et al.*, 2014; MARCONATO; BRANCO, 2014). O modelo em camadas facilita o uso de Arquiteturas Orientadas a Serviço em SANTs (RODRIGUES *et al.*, 2016) e a possibilidade de se identificar problemas devido ao isolamento proporcionado por uma arquitetura em camadas.

6.2 Limitações do Modelo de Arquitetura LARISSA

Como toda pesquisa, a aqui realizada também apresenta limitações. Uma vez que o trabalho realizado possui um prazo para ser finalizado não é possível que todas as possibilidades sejam abordadas e tratadas, fazendo-se assim necessário delimitar o escopo do trabalho.

- **Abrangência e detalhamento do modelo:** As principais limitações do presente trabalho, como citado anteriormente, referem-se abrangência e detalhamento do modelo de arquitetura, isto devido principalmente aos fatores inovação e ausência de padrões. A indústria e a academia lançam constantemente inovações para uso em SANTs. Neste sentido foi concebido um modelo de arquitetura que não pode ser considerado estático, mas sim **dinâmico** e **flexível** pois permite que alterações sejam realizadas e novidades tecnológicas inseridas. A inexistência de uma padronização mínima remete a uma larga escala de componentes a serem mapeados, ou seja, que necessitam de desenvolvimento de subcamadas com maior nível de abstração e detalhamento destes objetos.
- **O *profile* LARISSA:** Assim como o modelo de arquitetura, o *profile* apresenta limitações, uma vez que o mesmo reflete o modelo estendendo o comportamento e apresentando um detalhamento maior de certos estereótipos que dependem de manuais técnicos ou *experts* da área.
- **Validação do modelo de arquitetura:** A realização da validação plena do modelo de arquitetura implica na construção de um SANT completo, hardware e software. Visto a impossibilidade de implementar um projeto desta magnitude durante o período de realização deste projeto, optou-se, como descrito no [Capítulo 4](#), validar partes do modelo da arquitetura. Optou-se por partes das mais diferentes camadas. O segmento terrestre não foi validado na sua totalidade.
- **Cenários de Testes:** Os testes conduzidos demonstram a aplicabilidade e viabilidade do LARISSA em diferentes cenários, entretanto, o número de cenários possíveis de serem simulados está longe de ser esgotado. Sendo assim inúmeros outros cenários podem e devem ser explorados.

6.3 Dificuldades Encontradas

Como mencionado, essa pesquisa envolveu conhecimentos multidisciplinares, o que permitiu gerar resultados inovadores ao custo de lidar com diversas dificuldades, não só tecnológicas como também científicas, aqui elencadas:

- **Espalhamento dos conceitos:** uma vez que a pesquisa é interdisciplinar, os conceitos encontravam-se pulverizados na literatura e buscavam atender somente a área em questão.

Esse trabalho permitiu a integração de diferentes conceitos para a realização e obtenção de uma pesquisa sólida e com resultados que demonstram a eficiência da agregação de áreas. Foram abordados conceitos de diferentes áreas, cabendo destacar: sistemas embarcados, veículo inteligentes, engenharia de software;

- **Obtenção dos requisitos para compor o LARISSA:** apesar dos VANTs não serem veículos recentes, o seu uso tem se intensificado nos últimos anos e, sendo assim, os detalhes de sua concepção ainda não estão bem documentados na literatura. Muito se tem sobre sistemas de controle, hardware e sistemas de aviônica, mas pouco se tem sobre a documentação propriamente dita. Sendo assim, o levantamento dos requisitos para a concepção do modelo de arquitetura dependeu de consultas e entrevistas com profissionais especialistas na área, bem como em artigos pulverizados na literatura. A participação dos membros do grupo de pesquisa no qual esse projeto está inserido no Instituto Nacional de Ciência e Tecnologia em Sistemas Embarcados Críticos (INCT-SEC) auxiliou a superar essa dificuldade;
- **Integração das tecnologias e abordagens:** a integração das diferentes áreas, vista por meio do LARISSA, aparenta ser trivial. No entanto, a definição do modelo de arquitetura e sua validação, bem como a proposição do simulador AVENS com a integração de dois outros simuladores, um de voo (X-Plane) e outro de redes de computadores (OMNeT++), demandou muito tempo, além de todo o processo de implementação e validação, trabalhoso e desafiador, que apesar de ser tecnológico é necessário para demonstrar e apresentar a prova de conceito da validade do LARISSA, seu *profile*, do *larissamodel2simulation* e do AVENS;
- **Ausência de regras:** como ainda não foram estipuladas regras, no caso do Brasil, pela ANAC fica prejudicado o desenvolvimento de certas camadas.

6.4 Trabalhos Futuros

O objetivo do LARISSA é permitir o desenvolvimento de SANTs de forma a permitir uma certa padronização que auxilie o processo de certificação dessas aeronaves. Na sua presente versão o LARISSA permite que esse objetivo seja quase que plenamente atendido, de modo que ele pode ser estendido e novas funcionalidades podem ser incorporadas. Inúmeras são as possibilidades de extensão do LARISSA, sendo assim, sugerem-se os seguintes trabalhos a serem realizados que permitirão dar continuidade ao LARISSA:

- Gerar aplicações, tais como sistemas embarcados e de uso em estações de controle terrestre, modelados em UML, com o *profile* LARISSA, SysML¹ e *Modeling And Analysis Of Real-*

¹ <<http://www.omg.sysml.org/>>

Time Embedded Systems (MARTE)²;

- Estender os módulos de mobilidade do OMNeT++ para simular e representar de forma mais próxima do real os tipos de voo de VANT;
- Aperfeiçoar o *larissamodel2simulation* para gerar configurações de todas as camadas e subcamadas advindas do *profile* LARISSA;
- Desenvolver mecanismos colaborativos que possam manter o modelo de arquitetura e o *profile* LARISSA atualizados e abrangentes;
- Elaborar outros cenários e executar simulações do domínio SANT utilizando as ferramentas concebidas neste trabalho (incluem-se aqui diversos experimentos que estão sendo realizados e planejados em parceria com a ENAC como a concepção de um PAPARAZZI³ reprojetoado almejando melhor desempenho e melhores características de comunicação⁴);
- Aperfeiçoar e desenvolver outros módulos e *plugins* que permitam estender e aprimorar o AVENS, possibilitando testes e validações mais completas dos SANTs, por exemplo a de Navegação e Controle e a de Missão.
- Detalhar e implementar no domínio SANT e consequentemente no LARISSA, os níveis de criticidade do sistema.

6.5 Declaração de autoria

Eu confirmo que esta tese de doutorado não foi apresentada como parte dos requisitos para obtenção de um outro grau nesta ou em qualquer outra instituição de ensino e pesquisa. Afirmo ainda que ela é o resultado do meu próprio trabalho e todo o material de outras fontes foi plenamente e adequadamente reconhecido/referenciado. Pesquisas realizadas em colaboração estão claramente indicadas. Trechos desta tese, quer tenham sido publicados ou submetidos à apreciação dos conselhos editoriais de revistas, conferências e *workshops*, são apresentados a seguir de acordo com a lista de publicações. Minhas contribuições a cada publicação também são listadas.

² <<http://www.omg.org/spec/MARTE/>>

³ <<http://paparazziuav.org>>

⁴ Foi submetido um projeto entre o LSEC/ICMC em conjunto com a ENAC contando com o envolvimento da *DelairTech Company* com esse propósito. Neste projeto o candidato foi indicado como bolsista de pós-doutorado para conduzir o projeto.

6.6 Lista de publicações

6.6.1 Artigos publicados

- **MARCONATO, E. A.;** PIGATTO, D. F.; MAXA, J. A.; PINTO, A. S. R.; LARRIEU N.; BRANCO, K. R. L. J. C. . IEEE 802.11n vs. IEEE 802.15.4: a study on Communication QoS to provide Safe FANETs.

Evento: 2nd Workshop on Safety and Security of Intelligent Vehicles (SSIV 2016), 2016, Toulouse.

Nível de contribuição: Alto - o aluno de doutorado é o principal pesquisador e conduziu os trabalhos em conjunto com os demais colaboradores.

- **MARCONATO, E. A.;** RODRIGUES, D.; CHAVES, A. A. ; BRANCO, K. R. L. J. C. ; RAMDHANY, R. ; COULSON, G. . Unmanned Aerial Vehicles & Service-Oriented Architecture: LARISSA and Knowledge Based Framework's First Study.

Evento: The Tenth International Conference on Autonomic and Autonomous Systems (ICAS), 2014, Chamonix.

Nível de contribuição: Alto - o aluno de doutorado é o principal pesquisador e conduziu os trabalhos em conjunto com os demais colaboradores.

- **MARCONATO, E. A.;** PIGATTO, D. F. ; BRANCO, K. R. L. J. C. ; BRANCO, L. H. C. . LARISSA: Layered architecture model for interconnection of systems in UAS.

Evento: 2014 International Conference on Unmanned Aircraft Systems (ICUAS), 2014, Orlando.

Nível de contribuição: Alto - o aluno de doutorado é o principal pesquisador e conduziu os trabalhos em conjunto com os demais colaboradores.

- **MARCONATO, E. A.;** BRANCO, K. R. L. J. C. . Modelo de Arquitetura em Camadas para Interconexão de Sistemas em VANT.

Evento: II Workshop of Communication in Critical Embedded Systems (WoCCES), 2014, Florianópolis.

Nível de contribuição: Alto - o aluno de doutorado é o principal pesquisador e conduziu os trabalhos em conjunto com os demais colaboradores.

- **SILVA, N. B. F. ; MARCONATO, E. A. ;** BRANCO, K. R. L. J. C. . AVALON: definition and modeling of a vertical takeoff and landing UAV.

Journal: Journal of Physics. Conference Series (Online), v. 633, p. 012125, 2015.

Nível de contribuição: Médio - o aluno de doutorado auxiliou na estrutura do artigo, assim como em sua escrita uma vez que a aeronave apresentada no referido artigo foi desenvolvida seguindo o LARISSA.

- RODRIGUES, D.; PIRES, R. M.; ESTRELLA, J. C. ; **MARCONATO, E. A.**; JUNIOR, O. T. ; BRANCO, K. R. L. J. C. . Using SOA in Critical-Embedded Systems.

Evento: 4th IEEE Int'l Conference on Cyber, Physical and Social Computing (CPSCoM), 2011, Dalian.

Nível de contribuição: Baixo - o aluno de doutorado auxiliou na escrita do artigo e as contribuições no que se refere ao modelo de arquitetura presente no texto.

6.6.2 *Artigos aceitos para publicação*

- RODRIGUES, D.; PIRES, R. M.; **MARCONATO, E. A.**; AREIAS, C.; CUNHA, J.; BRANCO, K. R. L. J. C., VIEIRA, M. . Service-Oriented Architectures for a Flexible and Safe Use of Unmanned Aerial Vehicles.

Journal: ITS MAGAZINE.

Nível de contribuição: Médio - o aluno de doutorado auxiliou na estrutura do artigo, assim como em sua escrita uma vez que o parte do conteúdo apresentado no referido artigo demonstram a importância e viabilidade do uso do LARISSA na utilização de SOA em VANTs.

6.6.3 *Artigos submetidos*

- **MARCONATO, E. A.**; RODRIGUES, M.; PIRES, R. M.; PIGATTO, D. F.; QUERINO FILHO, L. C.; PINTO, A. R.; BRANCO, K. R. L. J. C. . AVENS – A Novel Flying Ad Hoc Network Simulator with Automatic Code Generation for Unmanned Aircraft System - Submetido.

Evento: Hawaii International Conference on System Sciences HICSS-50, 2017, Honolulu, Hawaii.

Nível de contribuição: Alto - o aluno de doutorado é o principal pesquisador e conduziu os trabalhos em conjunto com os demais colaboradores.

- **MARCONATO, E. A.**; PIGATTO, D. F.; BRANCO, L. H. C.; BRANCO, K. R. L. J. C. . LARISSA Profile: a UML profile for specifying the Layered Architecture Model for Interconnection of Systems in UAS - Submetido.

Journal: Journal of Intelligent and Robotic Systems (JINT)

Nível de contribuição: Alto - o aluno de doutorado é o principal pesquisador e conduziu os trabalhos em conjunto com os demais colaboradores.

REFERÊNCIAS

ANAC. **Manual de procedimentos - Certificação de projeto de tipo brasileiro - MPR-200**. [S.l.], 2010. Revisão 02. Citado 6 vezes nas páginas 36, 39, 117, 175, 177 e 178.

_____. **Requisitos de Aeronavegabilidade: Aeronaves de Asas Rotativas, Categoria Normal - RBAC 27**. [S.l.], 2013. Citado na página 36.

_____. **Requisitos de Aeronavegabilidade: Aviões Categoria Normal, Utilidade, Acrobática e Transporte - RBAC 23**. [S.l.], 2013. Citado na página 36.

_____. **Proposição de um Regulamento Especial para Sistema de Aeronaves Remotamente Pilotadas – RPAS e Emenda ao RBAC 67**. [S.l.], 2015. Citado na página 35.

_____. **Requisitos Gerais para Veículos Aéreos não Tripulados e Aeromodelos**. [S.l.], 2015. Citado 3 vezes nas páginas 35, 44 e 45.

ARDUPILOT, D. T. **Ardupilot**. 2011. Disponível em: <<http://plane.ardupilot.com>>. Acesso em: 07/10/2012. Citado na página 51.

ARMOUSH, A.; BECKSCHULZE, E.; KOWALEWSKI, S. Safety assessment of design patterns for safety-critical embedded systems. In: SEAA '09: PROCEEDINGS OF THE 2009 35TH EUROMICRO CONFERENCE ON SOFTWARE ENGINEERING AND ADVANCED APPLICATIONS. Washington, DC: IEEE Computer Society, 2009. p. 523–527. Citado na página 33.

ARMY, U. S. **U.S. Army Roadmap for UAS 2010-2035**. Alabama, 2010. Citado 4 vezes nas páginas 36, 47, 48 e 49.

AUSTIN, R. **Unmanned aircraft systems : UAVS design, development and deployment**. Chichester: Wiley, 2010. 332 p. Citado 9 vezes nas páginas 34, 41, 42, 43, 45, 46, 48, 49 e 50.

AUTHORITY, U. C. A. **Unmanned Aircraft System Operations in UK Airspace—Guidance**. [S.l.]: CAP722, 2012. Citado na página 36.

BARBER, K. S.; HOLT, J.; BAKER, G. Performance evaluation of domain reference architectures. In: PROC. OF THE 14TH INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING AND KNOWLEDGE ENGINEERING. [S.l.]: ACM Press, 2002. p. 225–232. Citado na página 56.

BASS, L.; CLEMENTS, P.; KAZMAN, R. **Software architecture in practice. The SEI Series in Software Engineering**. 2. ed. [S.l.]: Addison-Wesley Publishing Company, 2003. Citado na página 56.

BATORY, D.; COGLIANESE, L.; GOODWIN, M.; SHAFER, S. Creating reference architectures: an example from avionics. In: SSR '95: PROC. OF THE 1995 SYMPOSIUM ON SOFTWARE REUSABILITY. New York, NY: ACM Press, 1995. p. 27–37. Citado na página 56.

- BEKMEZCI, I.; SAHINGOZ, O. K.; TEMEL, S. Flying ad-hoc networks (fanets): A survey. v. 11 (3), p. 1254–1270, May 2013. Citado 2 vezes nas páginas 123 e 174.
- BIOLCHINI, J.; MIAN, P. G.; NATALI, A. C. C.; TRAVASSOS, G. H. Systematic review in software engineering. **System Engineering and Computer Science Department COPPE/UFRJ, Technical Report ES**, v. 679, n. 05, p. 45, 2005. Citado na página 199.
- BIOLCHINI, J. C. de A.; MIAN, P. G.; NATALI, A. C. C.; CONTE, T. U.; TRAVASSOS, G. H. Scientific research ontology to support systematic review in software engineering. **Advanced Engineering Informatics**, Elsevier, v. 21, n. 2, p. 133–151, 2007. Citado na página 200.
- BLAHA, M.; RUMBAUGH, J. **Modelagem e projetos baseados em objetos com uml2**. 2. ed. Rio de Janeiro: Elsevier, 2006. 496 p. Citado na página 92.
- BOOCH, G.; RUMBAUGH, J.; JACOBSON, I. **UML - Guia do usuário**. Rio de Janeiro: Elsevier, 2006. Citado 2 vezes nas páginas 91 e 92.
- BOUACHIR, O.; ABRASSART, A.; GARCIA, F.; LARRIEU, N. A mobility model for uav ad hoc network. In: **Unmanned Aircraft Systems (ICUAS), 2014 International Conference on**. [S.l.: s.n.], 2014. p. 383–388. Citado na página 165.
- BOUKERCHE, A.; OLIVEIRA, H. A. B. F.; NAKAMURA, E. F.; LOUREIRO, A. A. F. Vehicular ad hoc networks: A new challenge for localization-based systems. **Computer Communications**, v. 31, n. 12, p. 2838 – 2849, 2008. ISSN 0140-3664. Mobility Protocols for ITS/VANET. Citado na página 164.
- BRAGA, R.; BRANCO, K. R. L. J. C.; JR, O. T.; MASIERO, P. C.; NERIS, L. O.; BECKER, M. Prolices: An approach to develop product lines for safety-critical embedded systems. In: XXXVII LATIN-AMERICAN INFORMATICS CONFERENCE (CLEI XXXVII). Quito, 2011. p. 1–16. Citado na página 195.
- BRAGA, R. T. V.; BRANCO, K. R. J. C.; JR, O. T.; GIMENES, I. M. S. Evolving tiriba design towards a product line of small electric-powered uavs. In: I BRAZILIAN CONFERENCE ON CRITICAL EMBEDDED SYSTEMS. [S.l.], 2011. Citado na página 195.
- BRANCO, K. R. L. J. C. **Contribuições na Área de sistemas distribuídos e redes de computadores e suas aplicações em sistemas embarcados críticos**. São Carlos, SP, SP: [s.n.], 2012. Citado na página 34.
- BRANCO, K. R. L. J. C.; PELIZZONI, J. M.; NERIS, L. O.; TRINDADE, O.; OSÓRIO, F. S.; WOLF, D. F. Tiriba a new approach of uav based on model driven development and multiprocessors. In: IEEE INTERNATIONAL CONFERENCE ON ROBOTICS AND AUTOMATION - ICRA COMMUNICATIONS. Shangai: IEEE, 2011. p. 6584–6587. Citado na página 34.
- CAA. **Unmanned aircraft system operations in UK airspace – Guidance**. [S.l.], 2015. Citado na página 44.
- CHAMBERLAIN, T. **Learning OMNeT++**. [S.l.]: Packt Publishing Ltd, 2013. Citado na página 105.
- CHOI, H.; YEOM, K. An approach to software architecture evaluation with the 4+1 view model of architecture. In: SOFTWARE ENGINEERING CONFERENCE. Ninth Asia-Pacific, 2002. p. 286–293. Citado na página 93.

CRAIGHEAD, J.; MURPHY, R.; BURKE, J.; GOLDIEZ, B. A survey of commercial & open source unmanned vehicle simulators. In: IEEE. **Proceedings 2007 IEEE International Conference on Robotics and Automation**. [S.l.], 2007. p. 852–857. Citado na página 165.

DEFESA, M. da. Requisitos operacionais conjuntos (roc) para o sistema de veículo aéreo não-tripulado de inteligência, reconhecimento, vigilância e aquisição de alvos (vant/arp istar) das forças armadas. **Diário Oficial da União**, n. 127, p. 10, 2013. Citado na página 195.

DENG, Z.; MA, C.; ZHU, M. A reconfigurable flight control system architecture for small unmanned aerial vehicles. In: 2012 IEEE INTERNATIONAL SYSTEMS CONFERENCE (SYSCON). Vancouver: IEEE, 2012. p. 1–4. Citado 4 vezes nas páginas 63, 64, 65 e 66.

DOD. **OSD UAV Roadmap 2002-2027**. Washington,DC, 2002. 205 p. Citado na página 34.

_____. **Unmanned Aircraft Systems Roadmap 2005-2030**. Washington,DC, 2005. 212 p. Citado na página 34.

_____. **Unmanned Systems Roadmap 2007-2032**. Washington,DC, 2007. 187 p. Citado 2 vezes nas páginas 34 e 43.

_____. **Unmanned aircraft systems - federal actions needed to ensure safety and expand their potential uses within the national airspace system**. Washington,DC, 2008. Citado na página 34.

_____. **Unmanned Systems Integrated Roadmap FY2009-2034**. Washington,DC, 2009. 210 p. Citado 3 vezes nas páginas 34, 43 e 44.

_____. **Unmanned systems integrated roadmap: FY2013-2038**. Washington,DC, 2013. Citado na página 36.

DUDEK, G.; JENKIN, M. **Computational Principles of Mobile Robotics**. [S.l.]: Cambridge Press, 2000. Citado na página 41.

DUNN, W. R. Designing safety-critical computer systems. **Computer**, v. 36, n. 11, p. 40–46, 2003. Citado na página 33.

EASA. **Introduction of a regulatory framework for the operation of unmanned aircraft**. [S.l.], 2015. Citado na página 45.

ECLIPSE, F. **Acceleo**. 2014. Disponível em: <<https://eclipse.org/acceleo/support/>>. Acesso em: 13/01/2014. Citado na página 103.

_____. **The Eclipse Foundation**. 2014. Disponível em: <<http://www.eclipse.org/home/index.php>>. Acesso em: 12/01/2014. Citado na página 92.

_____. **Papyrus**. 2014. Disponível em: <<http://eclipse.org/papyrus/>>. Acesso em: 13/01/2014. Citado na página 92.

EICKELMANN, N. S.; RICHARDSON, D. J. An evaluation of software test environment architectures. In: PROC. OF THE 18TH INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING. Berlin, Germany: IEEE Computer Society Press, 1996. p. 353–364. Citado na página 56.

ENAC. **Paparazzi The Free Autopilot**. 2012. Disponível em: <<http://paparazzi.enac.fr>>. Acesso em: 02/02/2012. Citado 2 vezes nas páginas 149 e 167.

_____. **YAPA/v2.0**. 2012. Disponível em: <<http://wiki.paparazziuav.org/wiki/YAPA/v2.0>>. Acesso em: 05/02/2012. Citado na página 150.

FURLONG, K.; ERICKSON, R. **The Power of 802.15.4 and Ethernet**. [S.l.], 2011. Disponível em: <<https://www.lsr.com/white-papers/the-power-of-802-15-4-and-ethernet>>. Citado na página 140.

GRANT, T. Unifying planning and control using an ooda-based architecture. In: ANNUAL RESEARCH CONFERENCE OF THE SOUTH AFRICAN INSTITUTE OF COMPUTER SCIENTISTS AND INFORMATION TECHNOLOGISTS ON IT RESEARCH IN DEVELOPING COUNTRIES, REPUBLIC OF SOUTH AFRICA: SOUTH AFRICAN INSTITUTE FOR COMPUTER SCIENTISTS AND INFORMATION TECHNOLOGISTS. [S.l.], 2005. p. 159–170. Citado na página 55.

HERNANDES, E.; ZAMBONI, A.; FABBRI, S.; THOMMAZO, A. D. Using gqm and tam to evaluate start-a tool that supports systematic review. **CLEI Electronic Journal**, Centro Latinoamericano de Estudios en Informática, v. 15, n. 1, p. 3–3, 2012. Citado na página 199.

IEC, I. E. C. **IEC 61508 edition 2.0**. [S.l.], 2010. Citado na página 36.

ILARSLAN, M.; BAYRAKCEKEN, M. K.; ARISOY, A. Avionics system design of a mini vtol uav. **Aerospace and Electronic Systems Magazine, IEEE**, v. 26, n. 10, p. 35–40, 2011. Citado 2 vezes nas páginas 68 e 70.

INET. **INET Framework**. 2014. Disponível em: <<https://inet.omnetpp.org>>. Acesso em: 04/03/2014. Citado na página 107.

JIN, H.; LEE, S.; HAN, S.; JO, H.; KIM, D. Wip abstract: Challenges and strategies for exploiting integrated modular avionics on unmanned aerial vehicles. In: 2012 IEEE/ACM THIRD INTERNATIONAL CONFERENCE ON CYBER-PHYSICAL SYSTEMS. Beijing: IEEE Computer Society, 2012. p. 211. Citado 2 vezes nas páginas 72 e 73.

JUN, L.; ZHIQIANG, Y.; JIANZHONG, Q.; SHUKUAN, L. A role-based reference model for the service properties of service oriented architecture. In: IFITA '09: INTERNATIONAL FORUM ON INFORMATION TECHNOLOGY AND APPLICATIONS. [S.l.], 2009. p. 341–349. Citado na página 56.

KAYAYURT, B.; YAYLA, I.; YAPICI, A.; KUCUKOGUZ, C. Ground control station avionics software development in anka uav. In: 30TH DIGITAL AVIONICS SYSTEMS CONFERENCE (DASC). Seattle: IEEE/AIAA, 2011. p. 5B6–1 – 5B6–7. Citado na página 70.

KEELE, S. Guidelines for performing systematic literature reviews in software engineering. In: **Technical report, Ver. 2.3 EBSE Technical Report**. EBSE. [S.l.: s.n.], 2007. Citado na página 199.

KITCHENHAM, B. Procedures for performing systematic reviews. **Keele, UK, Keele University**, v. 33, n. 2004, p. 1–26, 2004. Citado na página 200.

KOULOHERIS, J. Future of system level design. In: PANEL DISCUSSION OF FIRST IEEE/ACM/IFIP INTERNATIONAL CONFERENCE ON HARDWARE/SOFTWARE CODESIGN AND SYSTEM SYNTHESIS. [S.l.], 2003. Citado na página 33.

KRUCHTEN, P. B. The 4+1 view model of architecture. **IEEE Software**, v. 12, n. 6, p. 42–50, Nov 1995. Citado na página 93.

KUMAR, S. P.; RAMAIAH, P. S.; KHANAA, V. Architectural patterns to design software safety based safety-critical systems. In: ICCCS 11: PROCEEDINGS OF THE 2011 INTERNATIONAL CONFERENCE ON COMMUNICATION, COMPUTING & SECURITY. New York: ACM, 2011. p. 620–623. Citado na página 33.

LAZIC, L.; VELASEVIC, D. Applying simulation and design of experiments to the embedded. **Software Testing, Verification Reliability**, v. 14, n. 4, p. 257–282, 2014. Citado na página 33.

LIZARRAGA, M. **Design, implementation and flight verification of a versatile and rapidly reconfigurable UAV GNC research platform**. Tese (Doctor of Philosophy in Computer Engineering) — University of California, Santa Cruz, 2009. Citado na página 142.

MARCONATO, E. A.; BRANCO, K. R. L. J. C. Modelo de arquitetura em camadas para interconexão de sistemas em vant. In: II WORKSHOP OF COMMUNICATION IN CRITICAL EMBEDDED SYSTEMS (WOCCEs). Florianópolis, 2014. p. 41–54. Citado 2 vezes nas páginas 40 e 178.

MARCONATO, E. A.; PIGATTO, D. F.; BRANCO, K. R. L. J. C.; BRANCO, L. H. C. Larissa: Layered architecture model for interconnection of systems in uas. In: 2014 INTERNATIONAL CONFERENCE ON UNMANNED AIRCRAFT SYSTEMS (ICUAS). Orlando, 2014. p. 20–31. Citado 2 vezes nas páginas 40 e 178.

MARCONATO, E. A.; PIGATTO, D. F.; MAXA, J. A.; PINTO, A. S. R.; N., L.; BRANCO, K. R. L. J. C. Ieee 802.11n vs. ieee 802.15.4: a study on communication qos to provide safe fanets. In: 2ND WORKSHOP ON SAFETY AND SECURITY OF INTELLIGENT VEHICLES (SSIV 2016). Toulouse, 2016. Citado 10 vezes nas páginas 39, 40, 133, 134, 135, 136, 137, 138, 139 e 178.

MARCONATO, E. A.; RODRIGUES, D.; CHAVES, A. A.; BRANCO, K. R. L. J. C.; RAMDHANY, R.; COULSON, G. Unmanned aerial vehicles & service-oriented architecture: Larissa and knowledge based framework's first study. In: THE TENTH INTERNATIONAL CONFERENCE ON AUTONOMIC AND AUTONOMOUS SYSTEMS (ICAS). Chamonix, 2014. p. 5–10. Citado 3 vezes nas páginas 40, 77 e 178.

MARCONATO, E. A.; RODRIGUES, M.; PIRES, R. M.; PIGATTO, D. F.; FILHO, L. C. Q.; PINTO, A. R.; BRANCO, K. R. L. J. C. Avens – a novel flying ad hoc network simulator with automatic code generation for unmanned aircraft system - submetido. In: HAWAII INTERNATIONAL CONFERENCE ON SYSTEM SCIENCES HICSS-50. Honolulu, Hawaii, 2017. Citado 5 vezes nas páginas 39, 171, 172, 173 e 178.

MATARIC, M. J. **The Robotics Primer**. [S.l.]: MIT Press, 2007. Citado na página 41.

MENDES, A. **Arquitetura de Software: Desenvolvimento orientado para arquitetura**. [S.l.]: Campus, 2002. Citado na página 56.

MIXIM. **MiXiM Framework**. 2014. Disponível em: <<http://mixim.sourceforge.net/>>. Acesso em: 15/01/2014. Citado 2 vezes nas páginas 107 e 171.

NATO/OTAN. **List of Current NATO Standards**. 2007. Disponível em: <<http://nsa.nato.int/nsa/nsdd/listpromulg.html>>. Acesso em: 15/10/2011. Citado na página 71.

NETO, J.; PAIXAO, R. da; RODRIGUES, L.; MOREIRA, E.; SANTOS, J. dos; ROSA, P. A surveillance task for a uav in a natural disaster scenario. In: 2012 IEEE INTERNATIONAL SYMPOSIUM ON INDUSTRIAL ELECTRONICS (ISIE). Hangzhou: IEEE, 2012. p. 1516–1522. Citado 2 vezes nas páginas 65 e 67.

NIST. **4D/RCS: Reference Model Architecture for Unmanned Vehicle Systems Version 2.0**. [S.l.], 2002. Citado 3 vezes nas páginas 48, 50 e 56.

NONAMI, K.; KENDOUL, F.; SUZUKI, S.; WANG, W.; NAKAZAWA, D. **Autonomous Flying Robots Unmanned Aerial Vehicles and Micro Aerial Vehicles**. [S.l.]: Springer, 2010. 329 p. Citado 5 vezes nas páginas 41, 43, 47, 50 e 51.

NS. **The Network Simulator**. 2014. Disponível em: <<https://www.nsnam.org>>. Acesso em: 01/03/2014. Citado na página 105.

OLIVEIRA, L. B. R. d. **Projeto arquitetural de sistemas robóticos orientados a serviços**. Tese (Doutorado em Ciências de Computação e Matemática Computacional) — Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos, 2015. Citado na página 56.

OLSON, L.; BURNS, L. A common architecture prototype for army tactical and fcs uavs. In: THE 24TH DIGITAL AVIONICS SYSTEMS CONFERENCE. DASC 2005. Washington: IEEE, 2005. p. 8.B.5–1 – 8.B.5–11. Citado 4 vezes nas páginas 57, 58, 59 e 60.

OMG, O. M. G. **Unified Modeling Language (UML)**. 2014. Disponível em: <<http://www.omg.org/spec/UML/>>. Acesso em: 05/03/2014. Citado na página 92.

_____. **OMG Systems Modeling Language (SysML)**. 2015. Disponível em: <<http://www.omg.org/spec/SysML/>>. Acesso em: 28/09/2015. Citado na página 93.

OMNET. **OMNeT++ documentation and tutorials**. 2014. Disponível em: <<https://omnetpp.org/doc/omnetpp/manual/>>. Acesso em: 15/03/2014. Citado 3 vezes nas páginas 39, 105 e 107.

OPENPILOT, T. **Openpilot**. 2010. Disponível em: <<http://www.openpilot.org>>. Acesso em: 28/08/2012. Citado na página 51.

PANICKER, S. **Applying DO178B for IV & V of Safety critical Software**. 2001. Disponível em: <<http://pt.scribd.com/doc/46780104/Applying-DO178B>>. Acesso em: 15/10/2012. Citado na página 71.

PASTOR, E.; LOPEZ, J.; ROYO, P. An embedded architecture for mission control of unmanned aerial vehicles. In: 9TH EUROMICRO CONFERENCE ON DIGITAL SYSTEM DESIGN ARCHITECTURES, METHODS AND TOOLS. Dubrovnik: IEEE Computer Society, 2006. p. 554–560. Citado 2 vezes nas páginas 42 e 57.

_____. Uav payload and mission control hardware / software architecture. **IEEE A&E Systems Magazine**, v. 22, p. 3–8, Junho 2007. Citado 9 vezes nas páginas 42, 45, 48, 49, 50, 60, 61, 62 e 63.

PERROCHON, L. A reference architecture for multi-author world-wide web servers. In: COCS '95: PROC. OF CONFERENCE ON ORGANIZATIONAL COMPUTING SYSTEMS. New York, NY: ACM Press, 1995. p. 197–205. Citado na página 56.

PIRES, R. M.; CHAVES, A. A.; BRANCO, K. R. Smart sensor protocol-a new standard for uav and payload integration. In: IEEE. **Unmanned Aircraft Systems (ICUAS), 2014 International Conference on**. [S.l.], 2014. p. 1300–1310. Citado na página [88](#).

PIRES, R. M.; RODRIGUES, D.; BRANCO, K. R. L. J. C.; TRINDADE, O. Mosa - mission oriented sensor array: A proposal. In: CLEI 11: PROCEEDINGS OF THE XXXVII CONFERENCIA LATINOAMERICANA DE INFORMÁTICA. [S.l.], 2011. p. 1309–1318. Citado na página [88](#).

PRISAZNUK, P. J. Integrated modular avionics. In: NATIONAL AEROSPACE AND ELECTRONICS CONFERENCE. NAECON 1992. [S.l.]: IEEE, 1992. p. 39–45. Citado na página [68](#).

REGLI, W. C.; MAYK, I.; DUGNA, C. J.; KOPENA, J. B.; LASS, R. N.; MODI, P. J.; MONGAN, W. M.; SALVAGE, J. K.; SULTANIK, E. A. Development and specification of a reference model for agent-based systems. **IEEE Transactions on Systems, Man, and Cybernetics - Part C: Applications and Reviews**, v. 39, n. 5, p. 572–596, 2009. Citado na página [56](#).

REN, W.; BEARD, R.; ATKINS, E. Information consensus in multivehicle cooperative control. **Control Systems, IEEE**, v. 27, n. 2, p. 71–82, April 2007. ISSN 1066-033X. Citado na página [165](#).

RIVER, I. W. Arinc 653 an avionics standard for safe, partitioned systems. In: IEEE-CS SEMINAR. [S.l.], 2008. Citado 4 vezes nas páginas [68](#), [69](#), [70](#) e [78](#).

RODRIGUES, D. **Arquitetura orientada a serviços para sistemas embarcados críticos complexos - um estudo de caso focado em aviônicos**. Tese (Qualificação de Doutorado em Ciências de Computação e Matemática Computacional) — Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos, 2013. Citado na página [77](#).

RODRIGUES, D.; PIRES, R.; MARCONATO, E. A.; AREIAS, C.; CUNHA, J.; BRANCO, K. R. L. J. C.; VIEIRA, M. Service-oriented architectures for a flexible and safe use of unmanned aerial vehicles - aceito. **ITS MAGAZINE**, 2016. Citado 3 vezes nas páginas [40](#), [77](#) e [178](#).

RODRIGUES, D.; PIRES, R. M.; ESTRELLA, J. C.; VIEIRA, M.; CORREA, M.; CAMARGO, J. B.; BRANCO, K. R. L. J. C.; TRINDADE, O. Application of soa in safety-critical embedded systems. **Communications in Computer and Information Science**, v. 206, p. 345–354, 2011. Citado na página [34](#).

RTCA. **DO-178C, Software Considerations in Airborne Systems and Equipment Certification**. [S.l.], 2012. Citado 2 vezes nas páginas [70](#) e [78](#).

SANDKUHL, K.; MESSER, B. Towards reference architectures for distributed groupware applications. In: 8TH EUROMICRO WORKSHOP ON PARALLEL AND DISTRIBUTED PROCESSING. Rhodes, Greece, 2000. Citado na página [56](#).

SILVA, N. B. F.; MARCONATO, E.; BRANCO, K. R. L. J. C. Avalon: definition and modeling of a vertical takeoff and landing uav. **Journal of Physics**, v. 633, p. 012125, 2015. Conference Series (Online). Citado 5 vezes nas páginas [39](#), [156](#), [158](#), [163](#) e [178](#).

SINGH, K.; VERMA, A. K. Experimental analysis of aodv, dsdv and olsr routing protocol for flying adhoc networks (fanets). In: IEEE. **Electrical, Computer and Communication Technologies (ICECCT), 2015 IEEE International Conference on**. [S.l.], 2015. p. 1–4. Citado na página [135](#).

SLUGS. **SLUGS - Santa Cruz Low-cost UAV GNC System**. Santa Cruz, 2012. Disponível em: <<https://slugsuav.soe.ucsc.edu/>>. Acesso em: 07/01/2012. Citado 2 vezes nas páginas 141 e 142.

SOMMER, C. **The open source vehicular network simulation framework**. 2014. Disponível em: <<http://veins.car2x.org/documentation/>>. Acesso em: 03/03/2014. Citado na página 164.

TANENBAUM, A. S.; WETHERALL, D. J. **Redes de Computadores**. 5. ed. São Paulo, SP Brasil: Pearson Education, 2011. 600 p. Citado na página 76.

TOEPKE, S. L. **Documentation, deployment and extension of a versatile and rapidly reconfigurable UAV GNC research platform**. Tese (Master of Science in Computer Engineering with an emphasis in Robotics & Control) — University of California, Santa Cruz, 2012. Citado 2 vezes nas páginas 51 e 141.

TRINDADE, O.; BARBOSA, L.; NERIS, L. O.; JORGE, L. A. C. A mission planner and navigation system for the arara project. In: ICAS - 23RD INTERNATIONAL CONGRESS OF AERONAUTICAL SCIENCES. Toronto: Optimage Ltd, 2002. p. 682.1 – 682.8. Citado na página 41.

TRINDADE, O.; BRANCO, K. R. L. J. C.; NERIS, L. O.; CHAVIER, L. F. C. Robôs aéreos. In: ROBÓTICA MÓVEL. [S.l.]: LCT, 2012. p. 461–478. Citado 3 vezes nas páginas 34, 42 e 43.

TRINDADE, O.; NERIS, L. O.; BARBOSA, L.; BRANCO, K. R. L. J. C. A layered approach to design autopilots. In: IEEE-ICIT 2010 INTERNATIONAL CONFERENCE ON INDUSTRIAL TECHNOLOGY. Viña del Mar: IEEE Press, 2010. p. 1395–1400. Citado 2 vezes nas páginas 34 e 41.

WANG, J.; XIN, M. Integrated optimal formation control of multiple unmanned aerial vehicles. **IEEE Trans. Contr. Sys. Techn.**, v. 21, n. 5, p. 1731–1744, 2013. Citado na página 165.

WATKINS, C. B.; WALTER, R. Transitioning from federated avionics architectures to integrated modular avionics. In: 26TH DIGITAL AVIONICS SYSTEMS CONFERENCE. DASC 07. Dallas: IEEE, 2007. p. 2.A.1–1 – 2.A.1–10. Citado na página 68.

WHITE, E. **Making Embedded Systems: Design Patterns for Great Software**. [S.l.]: O'Reilly Media, 2012. 328 p. Citado na página 33.

WU, X.; LING, H.; DONG, Y. On modeling and verifying of application protocols of ttcan in flight-control system with uppaal. In: INTERNATIONAL CONFERENCE ON EMBEDDED SOFTWARE AND SYSTEMS - ICESS '09. Hangzhou: IEEE Computer Society, 2009. p. 572–577. Citado na página 57.

X-PLANE. **X-Plane**. 2014. Disponível em: <http://www.xsquakebox.net/xpsdk/mediawiki/Main_Page>. Acesso em: 15/10/2014. Citado na página 165.

YANMAZ, E.; HAYAT, S.; SCHERER, J.; BETTSTETTER, C. Experimental performance analysis of two-hop aerial 802.11 networks. In: **Wireless Communications and Networking Conference (WCNC), 2014 IEEE**. [S.l.: s.n.], 2014. p. 3118–3123. Citado na página 165.

YANMAZ, E.; KUSCHNIG, R.; BETTSTETTER, C. Achieving air-ground communications in 802.11 networks with three-dimensional aerial mobility. In: **INFOCOM, 2013 Proceedings IEEE**. [S.l.: s.n.], 2013. p. 120–124. ISSN 0743-166X. Citado na página 165.

YEARBOOK, U. Unmanned aircraft systems—the global perspective 2011/2012. **Blyenburg & Co, June**, p. 1967–1709, 2011. Citado na página [36](#).

YI, Z.; CAI, W.; YUE, W. Adaptive safety critical middleware for distributed and embedded safety critical system. In: NCM 08: PROCEEDINGS OF THE 2008 FOURTH INTERNATIONAL CONFERENCE ON NETWORKED COMPUTING AND ADVANCED INFORMATION MANAGEMENT. Washington, DC: IEEE Computer Society, 2008. v. 1, p. 162–166. Citado na página [33](#).

ZDUN, U. Reengineering to the web: A reference architecture. In: SIXTH EUROPEAN CONFERENCE ON SOFTWARE MAINTENANCE AND REENGINEERING. Budapest, Hungary, 2002. Citado na página [56](#).

DOCUMENTO DE REQUISITOS

A.1 Introdução

Neste apêndice é apresentado um documento de requisitos básico para um VANT de asa fixa (o Tiriba), com propósito de auxiliar no levantamento das informações necessárias para a concepção do modelo de arquitetura LARISSA.

O levantamento dos requisitos de um VANT, associado a informações obtidas por outros profissionais e especialistas da área de veículos aéreos tripulados e não tripulados, além das informações de requisitos de artigos e documentações (BRAGA *et al.*, 2011a; BRAGA *et al.*, 2011b; DEFESA, 2013) que descrevem outros requisitos de diferentes VANTs que permitiu a concepção do LARISSA. Informações disponibilizadas por profissionais e materiais oriundos do projeto INCT-SEC¹ também auxiliaram neste processo.

A.1.1 *Objetivo do documento*

Este documento descreve os requisitos de um VANT (neste caso o Tiriba, um veículo aéreo de asa fixa desenvolvido no âmbito do INCT-SEC) que constitui um sistema crítico. Destina-se ao projetista e ao desenvolvedor desse sistema.

A.1.2 *Escopo do Produto: Projeto e desenvolvimento de um VANT capaz de realizar diversas aplicações dependendo do contexto.*

Veículos aéreos não tripulados (VANTs) ou do inglês UAVs (*Unmanned Aerial Vehicle*) possuem um enorme potencial de utilização ainda não explorado. Aplicações incluem levantamento topográfico, aerofotogrametria, monitoramento de tráfego, vigilância, planejamento urbano, inspeção de linhas de transmissão e dutos, entre várias outras utilizações possíveis. Desse

¹ <<http://www.inct-sec.icmc.usp.br/br/>>

modo o sistema está voltado para o sensoriamento remoto usando fotos e vídeos para captação de imagens aéreas tanto imagens fotográficas quanto de vídeo. Espera-se que a captura das imagens seja utilizada nas áreas de agricultura, monitoramento ambiental, monitoramento de defesa civil e aplicações de reconhecimento de terreno, e para aplicação na defesa.

A.1.3 Características do Usuário

A ideia básica é a não necessidade de conhecimento específico. Não há necessidade de treinamento específico, somente um treinamento básico. Não há necessidade por parte das pessoas que farão uso do VANT experiências e habilidades especiais nem mesmo treinamento intensivo. A interação do usuário com o VANT dar-se-á por intermédio da estação base (que pode ser desde um celular até mesmo um notebook). A interface da estação base é amigável e de fácil utilização.

A.2 Descrição Geral do Produto

A.2.1 Perspectiva do Produto

O sistema é composto de: Aeronave; câmera; bateria; carregador de bateria; software de definição de missão (esse software permite a programação de uma missão, o carregamento da missão na aeronave além de permitir verificar as baterias (e informar que as mesmas necessitam ser carregadas), permite o lançamento e o recolhimento da aeronave. A estação de base para voo interativo pode ser composta por um *notebook/tablet*, um celular e possui um software de interação.

A.2.2 Requisitos Funcionais

- F1- Aeronave deve permitir obtenção de imagens a partir de câmeras fotográficas ou de vídeo.
- F2 – Pontos de obtenção das imagens são informados na definição da missão.
- F3 – A aeronave deve permitir os modos de voo autônomo e/ou assistido.
- F4 – Câmera deve permitir posicionamento no eixo de *Roll*.
- F5 – Imagens obtidas dentro de precisão de posicionamento especificadas.
- F6 – As missões devem ser especificadas seguindo os seguintes requisitos: *Waypoints* e *Grids*.
- F7- Sistema deve possibilitar carga (*payload*) de 700gramas.
- F8 – Operações devem ser restritas a um polígono de voo previamente definido.

A.2.3 Requisitos Não Funcionais

- NF1 – Sistema deve apresentar baixo custo.
- NF2 – Permitir voos de duração de mais que 1 hora.
- NF3 – Velocidade de cruzeiro maior que 100 Km/h.
- NF4 – Lançamento manual por “estilingue”.
- NF5 – Peso inferior a 4 kilos.
- NF6 – Tempo de montagem da aeronave inferior a 20 minutos.
- NF7 – Sistema pode ser desmontado e carregado em uma mochila com dimensões que caibam a aeronave e os demais equipamentos.
- NF8 – Propulsão elétrica.
- NF9 – Pouso automático por meio de pára-quedas.
- NF10 – Pouso manual de barriga.
- NF11 – Estação de base baseada em *notebook/tablet* ou celular (*smartphone*).
- NF12 – Sistema de fácil utilização, simplesmente necessitando pouco ou nenhum treinamento ao usuário.
- NF13 – Navegação baseada em GPS.
- NF14 – Dispor de sensores inerciais para estabilização do voo (controle de voo)
- NF15 – Dispor de unidade barométrica para medição de velocidade, altitude e taxa de subida.

A.2.4 Requisitos de Desempenho

O desempenho do sistema é crítico durante a execução de uma missão, necessitando de respostas precisas e que os comandos sejam prontamente atendidos.

O desenvolvimento do sistema é baseado em metodologia com simulação e geração de código automático substituindo o programado de modo a ter uma menor interferência possível no sistema.

A.2.5 Requisitos de Segurança

A manutenção deve ser mínima sendo realizada por qualquer pessoa, não necessariamente um profissional especializado.

A.2.6 Não requisitos

Componentes de hardware baseado em componentes específicos (por exemplo, não requer componentes de especificação militar).

Primeira versão não exige procedimento que leve a certificação. Nenhum tipo de redundância se faz necessária no sistema.

REVISÃO SISTEMÁTICA

A forma de identificar, avaliar e interpretar as pesquisas relevantes de uma área em torno de uma questão científica é a chamada revisão sistemática. Resumir possíveis evidências de uma tecnologia, identificar lacunas para sugerir áreas que devem ser investigadas, esclarecer o estado da arte para o posicionamento de novas pesquisas e examinar resultados empíricos que comprovem ou contradizem uma determinada hipótese teórica ou que auxiliem na criação de novas hipóteses é possível por meio desta revisão (KEELE, 2007).

Esse processo foi realizado nesta tese com o auxílio da ferramenta *State of the Art through Systematic Review* (StArt) que foi desenvolvida pelo laboratório de Pesquisa em Engenharia de Software da Universidade Federal de São Carlos (UFSCAR) (HERNANDES *et al.*, 2012). Esta ferramenta oferece todo o suporte necessário para a aplicação do protocolo de revisão sistemática de uma maneira formal por meio de etapas bem definidas, proporcionando assim, a aplicação das recomendações presentes na literatura (KEELE, 2007), (BIOLCHINI *et al.*, 2005).

B.1 Planejamento

O planejamento é a primeira etapa da revisão sistemática, e nele são definidos os objetivos, as questões, as fontes da pesquisa e os critérios de seleção. Essas definições possibilitam seguir um protocolo pré-determinado para que o investigador não seja tendencioso.

B.1.1 Objetivos

Os principais objetivos motivadores da revisão sistemática são:

- Identificar as ferramentas, arquiteturas e modelos que possam auxiliar o desenvolvimento de SANTS;

- Identificar ferramentas modelos e arquiteturas que possam auxiliar a inserção dos VANTs no espaço aéreo;
- Identificar os modelos, arquiteturas e modelos de arquiteturas existentes para o desenvolvimento de veículos aéreos não tripulados de modo a auxiliar no seu processo de certificação facilitando assim sua inserção no espaço aéreo.

B.1.2 Questões

Durante o planejamento, a atividade mais importante é a formulação das questões, pois as mesmas estabelecem o foco de interesse da pesquisa (KITCHENHAM, 2004). As respostas dessas questões ajudam a realizar uma análise específica obter uma conclusão sobre o foco da pesquisa. Com base nos conceitos relacionados ao doutorado foram derivadas cinco questões que nortearam a pesquisa:

- Questão 1 - Quais as principais pesquisas sobre modelos de arquitetura de hardware e software para VANTs ou SANTs?
- Questão 2 - Quais níveis de hardware e software são especificados nos modelos de arquitetura encontrados na literatura?
- Questão 3 - Para quais categorias de VANTs/SANTs um modelo de arquitetura em camadas pode ser aplicado?
- Questão 4 - Um modelo de arquitetura em camadas, auxilia na concepção e certificação dos componentes do VANT/SANT?
- Questão 5 - Como é validado um modelo de arquitetura de hardware e software para VANTs/SANTs?

Para as questões construídas as seguintes especificidades devem ser consideradas (BIOLCHINI *et al.*, 2007):

- População: Serão observados projetos que envolvam esforços em apresentar arquiteturas, modelos e modelos de arquitetura que permitam a confecção de SANTs ou aeronaves com vistas à certificação e/ou padronização;
- Intervenção: No contexto da revisão sistemática serão analisadas modelos, arquiteturas e modelos de arquiteturas que levem em conta a especificação e o desenvolvimento de *profiles* de aeronaves, sejam elas tripuladas ou não tripuladas;
- Comparação: Avaliar e analisar diversas arquiteturas, modelos e modelos de arquitetura para a confecção de SANTs e VANTs;

- Resultado: Identificação dos modelos, arquiteturas e modelos de arquiteturas para SANTs, incluindo modelos de arquiteturas para aeronaves tripuladas.

B.1.3 Seleção das fontes

Outro elemento importante na revisão sistemática são as fontes que permitiram obter os trabalhos relevantes na área. Sendo assim, foi selecionado o método de consulta a bases de dados indexadas online, por permitirem fácil acesso aos trabalhos.

Foram definidas as fontes de fontes de busca segundo alguns critérios: Cobertura, Conteúdo Atualizado e Disponibilidade. Sobre a Cobertura, somente serão consideradas fontes de busca que retornem um número considerável de trabalhos. Mesmo retornando um grande número de trabalhos só serão considerados os que retornam trabalhos recentes sobre o tema (Conteúdo Atualizado). Quanto a Disponibilidade, serão utilizadas apenas fontes que permitam o acesso de forma íntegra ao trabalho. As seguintes bases foram escolhidas com base nos critérios:

- IEEE¹
- ACM²
- Scopus³
- Springer⁴

Outro critério estabelecido foi o idioma. Foram considerados nesta revisão sistemática artigos em inglês, por ser a língua adotada internacionalmente para a escrita de trabalhos científicos, e o português, para incluir trabalhos brasileiros, o que permite valorizar o trabalho desenvolvido no país.

B.1.4 Seleção dos estudos

No planejamento é necessário definir os critérios de seleção e os procedimentos de análise dos estudos, sendo esta a última etapa do planejamento.

Nesta etapa são definidos os critérios de inclusão e exclusão e os procedimentos para selecionar os estudos. A credibilidade da revisão sistemática é maior se esses critérios forem respeitados. O procedimento para selecionar os estudos define como a etapa de seleção de trabalhos e de análise de resultados é efetuada.

Os critérios de inclusão (CIs) definidos para responder às questões levantadas foram:

¹ <<http://ieeexplore.ieee.org/>>
² <<http://www.acm.org/>>
³ <<https://www.scopus.com/>>
⁴ <<http://www.springer.com/>>

- CI1 – estudos que apresentem modelos, arquitetura, modelos de arquitetura para confecção de SANTs;
- CI2 – estudos sobre modelos de referência para sistemas embarcados com foco em veículos aéreos, sejam eles tripulados ou não tripulados.

Tendo como base os objetivos da pesquisa, os seguintes critérios de exclusão (CEs) de trabalhos foram definidos:

- CE1 – estudos que não apresentem modelos, arquiteturas ou modelos de arquitetura para VANTs ou SANTs;
- CE2 – estudos em que não seja possível o acesso na íntegra do artigo;
- CE3 – estudos que estejam em idiomas diferentes do inglês ou do português;
- CE4 – estudos que não sejam completos (resumos).

As etapas do processo de execução da revisão sistemática são:

- A primeira etapa é a criação de strings de busca com base nas palavras-chave definidas anteriormente e com o uso dos operadores “AND”, utilizado para relacionar termos diferentes, e “OR”, utilizado para agrupar termos sinônimos ou que representem técnicas diferentes. Essas strings de busca criadas foram aplicadas nas máquinas de busca selecionadas pelo processo de revisão;
- A segunda etapa trata da seleção preliminar dos artigos por meio da leitura dos títulos e resumos. Nessa etapa devem ser identificados os estudos duplicados e a seleção é baseada nos critérios de inclusão e exclusão. Os estudos identificados como repetidos, ou seja, duplicados, não foram verificados em relação aos critérios de inclusão e exclusão. Os estudos analisados que se enquadraram em algum dos critérios de exclusão estabelecidos na revisão foram colocados em uma lista de estudos excluídos. Por fim, os estudos que se enquadraram em algum critério de inclusão foram inseridos em uma lista com o seu respectivo critério de inclusão. Esses estudos foram analisados de maneira mais detalhada e lidos por completo.
- Terceira etapa do processo é a de seleção final com a leitura completa dos artigos não-excluídos e a extração inicial das principais informações por meio da elaboração de um resumo.
- A quarta etapa trata da extração dos dados que abrangem modelos, arquiteturas, modelos de arquitetura para confecção de veículos aéreos sejam eles tripulados ou não tripulados.

B.2 Execução

O processo de revisão sistemática descrito neste capítulo foi conduzido no período de 07/03/2012 e 28/01/2013. Para o gerenciamento das referências bibliográficas utilizou-se as ferramentas Mendeley⁵ e StArt. Cabe ressaltar ainda que os resultados de busca foram obtidos quando realizados na rede da Universidade de São Paulo (USP), de modo que buscas fora da USP podem ocasionar resultados diferentes.

B.2.1 Construção da string de busca

A primeira etapa de criação de *strings* de busca foi realizada respeitando-se as especificações de cada base de dados, utilizando os operadores “AND” entre termos diferentes e “OR” para agrupar os sinônimos. As *strings* de busca criadas com os termos em inglês e português foram:

(UAV* OR UAS* OR "unmanned aerial vehicle" OR "unmanned aerial system") AND (architecture* OR "reference model") AND ((hardware AND software) OR (embedded AND system*))

(VANT* OR SANT* OR "Veículo aéreo não tripulado" OR "sistema aéreo não tripulado") AND (arquitetura* OR "modelo de referência") AND ((hardware AND software) OR (embarcado AND sistema*))

B.3 Seleção de estudos

A quantidade de artigos levantados em cada base de dados, pode ser observada na [Tabela 9](#)

Tabela 9 – Total de artigos - primeiro levantamento

Base	Quantidade
IEEE	76
ACM	18
Scopus	52
Springer	14
Total	160

Fonte: Elaborada pelo autor.

Após a obtenção dos levantamentos, estes foram analisados quanto a sua relevância, de forma a determinar se deveriam ou não ser considerados como estudos primários seguindo os critérios de inclusão e exclusão já apresentados anteriormente.

⁵ Maiores informações a respeito da ferramenta *Mendeley* podem ser encontradas em www.mendeley.com

Deste total de artigos, dois eram duplicados e após adotar os critérios a seleção preliminar resultou em 20 (vinte) e ao final da análise 08 (oito) artigos foram aceitos. A distribuição por ano destes artigos consta da [Tabela 10](#).

Tabela 10 – Distribuição por ano dos artigos aceitos na revisão sistemática

Ano	Quantidade
2005	1
2006	1
2007	1
2011	2
2012	3

Fonte: Elaborada pelo autor.

A quantidade de artigos que atendem cada requisito estão listados na [Tabela 11](#).

Tabela 11 – Requisitos atendidos

Requisito	Quantidade
Física do segmento aéreo	5
RTOS distribuída	2
Abstração do sistema	1
Monitoramento e controle	5
Navegação e serviços	8
Missão	7
Física do segmento terrestre	2
Estação de controle terrestre	7

Fonte: Elaborada pelo autor.

Por fim, a listagem dos artigos aceitos na revisão sistemática e abordada no [Capítulo 3](#) são relacionados no [Quadro 29](#) com as seguintes informações: identificador (ID), autores, título, ano de publicação.

Na última etapa da revisão sistemática foi produzida uma síntese discursiva para relatar os modelos, arquiteturas e modelos de arquiteturas encontrados, que foi apresentada em detalhes no [Capítulo 3](#).

A revisão foi reavaliada em Julho de 2016, não sendo encontrado resultados que alterassem os já apresentados anteriormente. Entretanto destaca-se que entre a execução dos primeiros levantamentos e este último, a IEEE⁶ alterou seu mecanismo de busca, impossibilitando a utilização da mesma *string* de busca.

⁶ <<http://ieeexplore.ieee.org/>>

Quadro 29 – Artigos aceitos na revisão sistemática

ID	Autores	Título	Ano
R1	Olson, L. and Burns, L.	A common architecture prototype for army tactical and FCS UAVS	2005
R2	Pastor, E. and Lopez, J. and Royo, P.	An Embedded Architecture for Mission Control of Unmanned Aerial Vehicles	2006
R3	Pastor, E. and Lopez, J. and Royo, P.	UAV Payload and Mission Control Hardware/Software Architecture	2007
R4	Deng, Z. and Ma, C. and Zhu M.	A Reconfigurable Flight Control System Architecture for Small Unmanned Aerial Vehicles	2012
R5	Neto, J.M.M. and da Paixao, R.A. and Rodrigues, L.R.L. and Moreira, E.M. and dos Santos, J.C.J. and Rosa, P.F.F.	A surveillance task for a UAV in a natural disaster scenario	2012
R6	Ilarslan, M. and Bayrakceken, M.K. and Arisoy, A.	Avionics system design of a mini VTOL UAV	2011
R7	Kayayurt, B. and Yayla, I. and Yapici, A. and Kucukoguz, C.	Ground control station avionics software development in ANKA UAV	2011
R8	Jin, H. and Lee, S. and Han, S. and Jo, H. and Kim, D.	WiP Abstract: Challenges and Strategies for Exploiting Integrated Modular Avionics on Unmanned Aerial Vehicles	2012

Fonte: Dados da pesquisa.

ESPECIFICAÇÃO DO *PROFILE* LARISSA

O *profile AirTrafficControl* é composto pelos estereótipos do [Quadro 30](#), sendo esta a subcamda que trata dos sensores embarcados no VANT que auxiliam o controle de tráfego aéreo a localizar e monitorar as aeronaves. Para os estereótipo *Transponder* as principais *tagged values* são *mode*, que especifica o modo de funcionamento do mesmo (A, C ou S) e *code*, utilizado pelo controle de tráfego aéreo, para identificar cada aeronave; para o estereótipo *ADS_B* tem-se a *tagged value service*, podendo ser setada para TIS_B, FIS_B ou Multilink. A notação para este *profile* é *larissa::UAS::Aerial::Physical::AvionicsElectronics::Sensor:: AirTrafficControl*.

Quadro 30 – *AirTrafficControl Profile*

Nº	Camada	Artefato
1.1.2.1.3.1	<i>Transponder</i>	Estereótipo
1.1.2.1.3.2	<i>ADS_B</i>	Estereótipo
1.1.2.1.3.3	<i>CPDLC</i>	Estereótipo

Fonte: Elaborada pelo autor.

O estereótipo *Stormscope* ([Quadro 31](#)) do *profile WeatherMonitoringForecasting* especifica o sensor utilizado nas aeronaves com a finalidade de verificar e monitorar as condições climáticas, como por exemplo a formação de tempestades. As *tagged values range*, *mode* e *view* compõem este estereótipo. A notação para este *profile* é *larissa::UAS::Aerial::Physical::AvionicsElectronics::Sensor::WeatherMonitoringForecasting*.

Quadro 31 – *WeatherMonitoringForecasting Profile*

Nº	Camada	Artefato
1.1.2.1.4.1	<i>Stormscope</i>	Estereótipo

Fonte: Elaborada pelo autor.

Ainda especificando o *profile AvionicsElectronics*, tem-se o *profile ServoAndActuator*, [Quadro 32](#), que especifica os dispositivos responsáveis pelo acionamento dos componentes

móveis de uma aeronave tais como *aileron*s, *flaps*, rotores, estabilizadores e lemes. Os estereótipos *Servo* e *Actuator* compõem o *profile*. Para ambos, as *tagged values torque* e *speed* especificam, respectivamente, a força que podem exercer e a velocidade de acionamento. Além destas os *Servos* do tipo elétricos possuem a *tagged value voltage* a qual especifica a voltagem ideal utilizada por este. O estereótipo *Actuator* possui a *tagged value system*, relativa ao tipo de acionamento do mesmo, podendo ser: *Mechanical*, *HydroMechanical* ou *FlyByWire*. A notação para este *profile* é *larissa::UAS::Aerial::Physical::AvionicsElectronics::ServoAndActuator*.

Quadro 32 – *ServoAndActuator Profile*

Nº	Camada	Artefato
1.1.2.2.1	<i>Servo</i>	Estereótipo
1.1.2.2.2	<i>Actuator</i>	Estereótipo

Fonte: Elaborada pelo autor.

O *profile Infrastructure*, carrega a notação *larissa::UAS::Aerial::Physical::AvionicsElectronics::Infrastructure* e é composto pelos estereótipos *ElectricalWiring* e *ElectricalConnector*, conforme listado no [Quadro 33](#). Estes tratam da fiação elétrica e conectores utilizados. As *tagged values length* e *thickness* que definem respectivamente, o comprimento e espessura do fio e pertencem ao estereótipo *ElectricalWiring*. As *tagged values model* e *type* o estereótipo *ElectricalConnector* são utilizados para definir o modelo (Molex, BNC, etc.) e o tipo (Macho, Fêmea ou nenhum) respectivamente,.

Quadro 33 – *Infrastructure Profile*

Nº	Camada	Artefato
1.1.2.3.1	<i>ElectricalWiring</i>	Estereótipo
1.1.2.3.2	<i>ElectricalConnector</i>	Estereótipo

Fonte: Elaborada pelo autor.

O estereótipo *Antennas*, listado no [Quadro 34](#) e pertencente ao *profile Antenna larissa::UAS::Aerial::Physical::AvionicsElectronics::Antenna*, especifica dados das antenas possíveis de serem utilizadas, seja no VANT ou na estação de controle terrestre. Suas *tagged values band* e *tracking* são utilizadas para especificar a banda (*band*) de transmissão/recepção utilizada no *link* de comunicação, tais como ISM, LEO, S, GEO, VHF e UHF. E o *tracking* especifica se antena é do tipo estacionária ou realiza rastreamento, apontando para aeronave, nesse caso podendo especificar se o rastreamento é mecânico ou elétrico.

Quadro 34 – *Antenna Profile*

Nº	Camada	Artefato
1.1.2.4.1	<i>Antennas</i>	Estereótipo

Fonte: Elaborada pelo autor.

Já o *profile Communications* possui duas subcamadas que originaram os *profiles AirToGround* e *AirToAir*, listados no [Quadro 35](#) e individualmente detalhados nos [Quadro 36](#) e [Quadro 39](#).

Quadro 35 – *Communications Profile*

Nº	Camada	Artefato
1.1.2.5.1	<i>AirToGround</i>	<i>profile</i>
1.1.2.5.2	<i>AirToAir</i>	<i>profile</i>

Fonte: Elaborada pelo autor.

O *profile AirToGround*, conforme o [Quadro 36](#), possui estereótipos e *profiles*. Para utilizar os estereótipos a notação é *larissa::UAS::Aerial::Physical::AvionicsElectronics::Communications::AirToGround*. Os *profiles* são detalhados nos [Quadro 37](#) e [Quadro 38](#). O estereótipo *WirelessDevice* especifica os dispositivos de rede sem fio utilizados em outras camadas onde o mesmos são estendidos. Suas *tagged values* são *standard*, que permite ao desenvolvedor escolher o padrão da camada física de rede como por exemplo o IEEE 802.11 ou IEEE 802.15.4, a frequência de transmissão/recepção (433Mhz, 868Mhz, 915Mhz, 2.4Ghz ou 5Ghz) - *frequency*, a potência de transmissão (*rfPower*), o canal (*channel*) e a sensibilidade de recepção (*sensitivity*). O estereótipo *AirTrafficControl* é utilizado para *link* de dados de controle de tráfego aéreo.

Quadro 36 – *AirToGround Profile*

Nº	Camada	Artefato
1.1.2.5.1.1	<i>WirelessDevice</i>	estereótipo
1.1.2.5.1.2	<i>AirTrafficControl</i>	estereótipo
1.1.2.5.1.3	<i>TelecommandTelemetry</i>	<i>profile</i>
1.1.2.5.1.4	<i>DataRelay</i>	<i>profile</i>

Fonte: Elaborada pelo autor.

O *profile TelecommandTelemetry* é composto pelo estereótipo *RadioModem*, [Quadro 37](#), o qual estende o estereótipo *WirelessDevice* utilizado para estabelecer o *link* de comunicação com a estação de controle terrestre e realizar a transmissão de dados de sensores, telemetria da aeronave para o solo e comandos de navegação, alterações de plano de voo e missão do solo para a aeronave. A notação para este *profile* é *larissa::UAS::Aerial::Physical::AvionicsElectronics::Communications::AirToGround::TelecommandTelemetry*.

Quadro 37 – *TelecommandTelemetry Profile*

Nº	Camada	Artefato
1.1.2.5.1.3.1	<i>RadioModem</i>	Estereótipo

Fonte: Elaborada pelo autor.

Assim como o *profile TelecommAndTelemetry*, o *profile DataRelay* estende o estereótipo *WirelessDevice* gerando o estereótipo *RelayDevice* listado no [Quadro 38](#). Ele é responsável

pelos *links* de comunicação utilizados para retransmitir dados de outros VANTs que não estão ao alcance de uma Estação de Controle Terrestre ou de um VANT para outro se também estiverem fora do alcance um do outro e necessitarem de um ponto de retransmissão. A notação para este *profile* é *larissa::UAS::Aerial::Physical::AvionicsElectronics::Communications::AirToGround::DataRelay*.

Quadro 38 – *DataRelay Profile*

Nº	Camada	Artefato
1.1.2.5.1.5.1	<i>RelayDevice</i>	Estereótipo

Fonte: Elaborada pelo autor.

O estereótipo *Coordination* do *profile AirToAir*, que consta do [Quadro 39](#), especifica os VANTs que podem ser Coordenadores de tráfego aéreo, de esquadrão ou nenhum deles. A *tagged value type* é utilizada para este fim. A notação para este *profile* é *larissa::UAS::Aerial::Physical::AvionicsElectronics::Communications::AirToAir*.

Quadro 39 – *AirToAir Profile*

Nº	Camada	Artefato
1.1.2.5.2.1	<i>Coordination</i>	Estereótipo

Fonte: Elaborada pelo autor.

Profile compartilhado tanto pelo segmento Aéreo quanto pelo Terrestre, o *Controller Profile*, [Quadro 40](#), é composto de estereótipos utilizados para modelar artefatos computacionais presentes em VANTs tais como controladoras de voo e processadores de missão e, modelar os artefatos computacionais presentes nas Estações de Controle Terrestre. As Estações de Controle Terrestre podem ser um simples computador ou uma rede de servidores, onde uma máquina desempenha uma tarefa específica ou funciona como contingência em caso de falha da máquina principal.

O estereótipo *Board* é utilizado para definir a aplicação principal do sistema computacional, a *tagged value boardType* é a responsável por esta funcionalidade, sendo que o desenvolvedor pode escolher entre controladora de voo, de missão, servidor, entre outras. O estereótipo *Memory* responde pelos tipos de memórias existentes na Controladora, sendo as *tagged values memoryType e size*, responsáveis, respectivamente, pelo tipo de memória (RAM, ROM, *Flash*) e a quantidade das mesmas.

O estereótipo *MicroController* especifica o processador utilizado na controladora, as *tagged values maker, speed e model* respondem pelo fabricante, velocidade e modelo do processador, respectivamente. A notação para este *profile* é *larissa::UAS::Aerial::Physical::AvionicsElectronics::Controller*.

Quadro 40 – *Controller Profile*

Nº	Camada	Artefato
1.1.2.6.1	<i>Board</i>	Estereótipo
1.1.2.6.2	<i>Memory</i>	Estereótipo
1.1.2.6.3	<i>MicroController</i>	Estereótipo

Fonte: Elaborada pelo autor.

Energy é o *profile* que representa a subcamada de Energia da camada Física, sendo composta pelas subcamadas *Propulsion*, *OnBoardPowerGenaration* e *Fuel* conforme ilustrado no [Quadro 41](#)

Quadro 41 – *Energy Profile*

Nº	Camada	Artefato
1.1.3.1	<i>Propulsion</i>	<i>profile</i>
1.1.3.2	<i>OnBoardPowerGenaration</i>	<i>profile</i>
1.1.3.3	<i>PowerSupplyAndConditioning</i>	<i>profile</i>
1.1.3.4	<i>Fuel</i>	<i>profile</i>

Fonte: Elaborada pelo autor.

O *profile Propulsion* tem como finalidade classificar as aeronaves quanto ao tipo de propulsão utilizada, sendo esta classificação representada pelos estereótipos *Reaction*, onde o princípio utilizado para a aeronave se locomover com seus próprios meios é o empuxo gerado pelos motores a reação comumente conhecidas como turbinas a jato. A *tagged value type* é utilizada para identificar se o motor é uma turbina a gás ou turbo-hélice. O estereótipo *Propoller* é utilizado para representar as aeronaves que utilizam a tração das hélices para se locomover e a *tagged value type* do estereótipo pode ser preenchida com *InternalCombustion*, *ElectricMotor* ou *Turboprop*, os quais representam motores de combustão interna, motores elétricos ou turbo-hélice.

No [Quadro 42](#) são listados os componentes deste *profile* e sua notação é *larissa::UAS::Aerial::Physical::Energy::Propulsion*.

Quadro 42 – *Propulsion Profile*

Nº	Camada	Artefato
1.1.3.1.1	<i>Reaction</i>	Estereótipo
1.1.3.1.2	<i>Propoller</i>	Estereótipo

Fonte: Elaborada pelo autor.

VANTs de grande porte, acima de 150Kg, podem embarcar sistemas de geração de energia tais como *Auxiliary Power Unit* (APU) - Unidade auxiliar de energia e o *profile OnBoardPowerGenaration* especifica o estereótipo *Generator*, [Quadro 43](#), que por meio da *tagged*

value type indica a finalidade do gerador: *Electricity*, para alimentar diretamente os dispositivos ou recarregar baterias; *SolarCell*, células solares que utilizam a energia solar para recarregar as baterias e; *FuelCell*, células de combustível. A notação para este *profile* é *larissa::UAS::Aerial::Physical::Energy::OnBoardPowerGenaration*.

Quadro 43 – *OnBoardPowerGenaration Profile*

Nº	Camada	Artefato
1.1.3.2.1	<i>Generator</i>	Estereótipo

Fonte: Elaborada pelo autor.

A subcamada que especifica a fonte de alimentação e acondicionamento da mesma no VANT é aqui representada pelo *profile PowerSupplyAndConditioning*. Este *profile* é composto pelos estereótipos *PowerSupply* e *Conditioning*, conforme Quadro 44. A notação para este *profile* é *larissa::UAS::Aerial::Physical::Energy::PowerSupplyAndConditioning*.

Quadro 44 – *PowerSupplyAndConditioning Profile*

Nº	Camada	Artefato
1.1.3.3.1	<i>PowerSupply</i>	Estereótipo
1.1.3.3.2	<i>Conditioning</i>	Estereótipo

Fonte: Elaborada pelo autor.

O Quadro 45 lista a composição do *profile Fuel* composto pelos estereótipos *AvGas* - Gasolina de Aviação, *Heavy* - Querosene e *Ethanol* - Etanol, os quais especificam as características ideais dos combustíveis utilizado em aeronaves com motores de combustão interna ou reação, e pelo *profile - Electricity* subcamada com estereótipos detalhados. A notação para este *profile* é *larissa::UAS::Aerial::Physical::Energy::Fuel*.

Quadro 45 – *Fuel Profile*

Nº	Camada	Artefato
1.1.3.4.1	<i>Electricity</i>	<i>profile</i>
1.1.3.4.2	<i>AvGas</i>	Estereótipo
1.1.3.4.3	<i>Heavy</i>	Estereótipo
1.1.3.4.4	<i>Ethanol</i>	Estereótipo

Fonte: Elaborada pelo autor.

O *profile Electricity* e seus estereótipos, listados no Quadro 46, especificam os objetos que fornecem energia elétrica. O estereótipo *battery* conta com *tagged values* que indicam a capacidade nominal em voltagem (*voltage*) e amperagem (*ampere*), número de células (*numCells*) que a compõe e o seu material químico: *LeadAcid* - Chumbo-Ácido, *NiCd* - Níquel-Cadmio, *LiPo* - Lítio-Polímero, *LiFe* - Lítio-Ferro, *LiIon* - Lítio-Ion ou *NiMH* - Níquel-Metal. Em todos os estereótipos tem-se um método que retorna sua voltagem atual. A notação para este *profile* é *larissa::UAS::Aerial::Physical::Energy::Fuel::Electricity*.

Quadro 46 – *Electricity Profile*

Nº	Camada	Artefato
1.1.3.4.1.1	<i>Battery</i>	Estereótipo
1.1.3.4.1.2	<i>FuelCell</i>	Estereótipo
1.1.3.4.1.3	<i>SolarCell</i>	Estereótipo

Fonte: Elaborada pelo autor.

O *profile AuxiliarySystems*, composto pelos estereótipos *TakeoffGear* e *LandingRecoveryGear*, listado no Quadro 47, especifica os meios pelos quais a aeronave pode alçar voo e retornar ao solo. O estereótipo *TakeoffGear* trata dos dispositivos utilizados para decolagem, tais como pneus, skis, catapultas, veículos de lançamento, entre outros. A recuperação da aeronave é representada pelo estereótipo *LandingRecoveryGear* e o desenvolvedor pode escolher entre pneus, skis, rede de recuperação, *air bag*, etc. A notação para este *profile* é *larissa::UAS::Aerial::Physical::AuxiliarySystems*.

Quadro 47 – *AuxiliarySystems Profile*

Nº	Camada	Artefato
1.1.4.1	<i>TakeoffGear</i>	Estereótipo
1.1.4.2	<i>LandingRecoveryGear</i>	Estereótipo

Fonte: Elaborada pelo autor.

Uma outra camada, que se encontra no mesmo nível da Física, representada com o *profile DistributedRTOS*, trata do sistema operacional de tempo real. Seus estereótipos *Driver* e *Network*, conforme Quadro 48, especificam as APIs que compõe esta camada sendo que o estereótipo *Driver* trata das APIs que possuem as especificações para realizar a troca de informações (mensagens) com os componentes que compõem a camada Física e as camadas superiores. Devido à especificidade dos dispositivos de rede, tais como segurança, eles são tratados pelo estereótipo *Network*. A notação para este *profile* é *larissa::UAS::Aerial::DistributedRTOS*.

Quadro 48 – *DistributedRTOS Profile*

Nº	Camada	Artefato
1.2.1	<i>Diver</i>	Estereótipo
1.2.2	<i>Network</i>	Estereótipo

Fonte: Elaborada pelo autor.

A camada de Monitoramento e Controle é representada com o *profile MonitoringAndControl* e seus estereótipos, os quais constam do Quadro 49. O estereótipo *FlightControl* especifica os componentes responsáveis pelo controle de voo, tais como comandos básicos de decolagem e pouso automáticos. O tratamento de emergência, situações que ocorrem durante a execução da missão e não estão previstas no plano de voo, são especificadas aplicando-se o estereótipo *EmergencyHandling*. Em aeronaves que possuem componentes redundantes o gerenciamento deste

sistema é especificado com o estereótipo *RedundancyHandling*. O gerenciamento de energia da aeronave, que mantém seu conjunto motopropulsor funcionando mantendo esta voando e o sistema de energia que mantém os demais componentes funcionando, tais como controladoras de voo, processadores de missão, *links* de comunicação e funcionando do *payload*, são especificados pelo uso do estereótipo *EnergyManagement*. A ciência de aeronavegabilidade, representada com o estereótipo *AirworthinessAwareness*, especifica o tratamento dos sensores embarcados que realizam a leitura de eventos normalmente perceptíveis por seres humanos, tais como trepidação, fumaça. Sua notação é *larissa::UAS::Aerial::MonitoringAndControl*.

Quadro 49 – *MonitoringAndControl Profile*

Nº	Camada	Artefato
1.3.1	<i>FlightControl</i>	Estereótipo
1.3.2	<i>EmergencyHandling</i>	Estereótipo
1.3.3	<i>RedundancyHandling</i>	Estereótipo
1.3.4	<i>EnergyManagement</i>	Estereótipo
1.3.5	<i>AirworthinessAwareness</i>	Estereótipo

Fonte: Elaborada pelo autor.

O *profile NavigationAndServices* e seus estereótipos listados no [Quadro 50](#) representam a camada de Navegação e Serviços. O estereótipo *AirTrafficCoordination* especifica a coordenação de tráfego aéreo ocupado pela aeronave. O controle de trajetória de voo (*FlightPathControl*) especifica os *waypoints* e *grids* de uma determinada missão. O estereótipo *GeoPoliticalAwareness*, subcamada Ciência Geopolítica, especifica os limites de voo que o VANT não pode ultrapassar impedindo assim que acidentalmente ele sobrevoe áreas restritas ou avance sobre a fronteira de outros país. Os serviços que auxiliam na navegação de aeronave são especificados com o estereótipo *Server*, onde pode-se pré-definir os tipos de servidores utilizados. A notação utilizada é *larissa::UAS::Aerial::NavigationAndServices*.

Quadro 50 – *NavigationAndServices Profile*

Nº	Camada	Artefato
1.4.1	<i>AirTrafficCoordination</i>	Estereótipo
1.4.2	<i>FlightPathControl</i>	Estereótipo
1.4.3	<i>GeoPoliticalAwareness</i>	Estereótipo
1.4.4	<i>Server</i>	Estereótipo

Fonte: Elaborada pelo autor.

A camada Missão foi modelada com o *profile Mission* e seus estereótipos, listados no [Quadro 51](#). Estes estereótipos especificam desde o acesso ao dispositivo que verifica se todos os pré-requisitos para execução de uma missão são atendidos pela configuração do VANT (*SSI* e *MOSA*), passando pelo Controle Automático (*AutomaticControl*), o qual estabelece que uma missão deve começar, parar, ser retomada ou ir para um próximo ponto. O estereótipo

RAWControl especifica os dados sem tratamento, ou seja, os dados coletados e que não necessitam de processamento interno na aeronave, podendo ser transmitidos para a Estação de Controle Terrestre e lá processados. A notação utilizada é *larissa::UAS::Aerial::Mission*.

Quadro 51 – *Mission Profile*

Nº	Camada	Artefato
1.5.1	<i>SSI</i>	Estereótipo
1.5.2	<i>MOSA</i>	Estereótipo
1.5.3	<i>AutomaticControl</i>	Estereótipo
1.5.4	<i>RAWControl</i>	Estereótipo

Fonte: Elaborada pelo autor.

CÓDIGO FONTE

LARISSAMODEL2SIMULATION

A parte essencial do *larissamodel2simulation* é composta por cinco arquivos a saber: Activator.java [Código-fonte 5](#), Generate.java [Código-fonte 6](#), generate.mtl [Código-fonte 7](#), UMLservices.java [Código-fonte 8](#) e umlservices.mtl [Código-fonte 9](#).

Código-fonte 5: Activator.java

```
1 package larissamodel2simulation;
2
3 import org.eclipse.core.runtime.Plugin;
4 import org.osgi.framework.BundleContext;
5
6 /**
7  * The activator class controls the plug-in life cycle.
8  */
9 public class Activator extends Plugin {
10     /**
11      * The plug-in ID.
12      */
13     public static final String PLUGIN_ID = "larissamodel2simulation";
14
15     /**
16      * The shared instance.
17      */
18     private static Activator plugin;
19     /**
20      * The constructor.
21      */
22     public Activator() {
23     }
```

```

24     /**
25      * {@inheritDoc}
26      *
27      * @see org.eclipse.core.runtime.Plugin#start(org.osgi.framework.
BundleContext)
28      */
29     public void start(BundleContext context) throws Exception {
30         super.start(context);
31         plugin = this;
32     }
33     /**
34      * {@inheritDoc}
35      *
36      * @see org.eclipse.core.runtime.Plugin#stop(org.osgi.framework.
BundleContext)
37      */
38     public void stop(BundleContext context) throws Exception {
39         plugin = null;
40         super.stop(context);
41     }
42     /**
43      * Returns the shared instance.
44      *
45      * @return the shared instance
46      */
47     public static Activator getDefault() {
48         return plugin;
49     }
50 }

```

Código-fonte 6: Generate.java

```

1 package larissamodel2simulation.main;
2
3 import java.io.File;
4 import java.io.IOException;
5 import java.util.ArrayList;
6 import java.util.List;
7
8 import org.eclipse.acceleo.engine.event.
IAccelerioTextGenerationListener;
9 import org.eclipse.acceleo.engine.generation.strategy.
IAccelerioGenerationStrategy;
10 import org.eclipse.acceleo.engine.service.AbstractAccelerioGenerator;
11 import org.eclipse.emf.common.util.BasicMonitor;
12 import org.eclipse.emf.common.util.Monitor;
13 import org.eclipse.emf.common.util.URI;

```

```

14 import org.eclipse.emf.ecore.EObject;
15 import org.eclipse.emf.ecore.resource.ResourceSet;
16
17 /**
18  * Entry point of the 'Generate' generation module.
19  *
20  * @generated
21  */
22 public class Generate extends AbstractAcceleioGenerator {
23     /**
24      * The name of the module.
25      *
26      * @generated
27      */
28     public static final String MODULE_FILE_NAME = "/
larissamodel2simulation/main/generate";
29
30     /**
31      * The name of the templates that are to be generated.
32      *
33      * @generated
34      */
35     public static final String[] TEMPLATE_NAMES = { "generateElement"
};
36
37     /**
38      * The list of properties files from the launch parameters (
Launch configuration).
39      *
40      * @generated
41      */
42     private List<String> propertiesFiles = new ArrayList<String>();
43     /**
44      * Allows the public constructor to be used. Note that a
generator created
45      * this way cannot be used to launch generations before one of
46      * {@link #initialize(EObject, File, List)} or
47      * {@link #initialize(URI, File, List)} is called.
48      * <p>
49      * The main reason for this constructor is to allow clients of
this
50      * generation to call it from another Java file, as it allows for
the
51      * retrieval of {@link #getProperties()} and
52      * {@link #getGenerationListeners()}.
53      * </p>
54      *

```

```

55     * @generated
56     */
57     public Generate() {
58         // Empty implementation
59     }
60     /**
61     * This allows clients to instantiates a generator with all
        required information.
62     *
63     * @param modelURI
64     *         URI where the model on which this generator will be
        used is located.
65     * @param targetFolder
66     *         This will be used as the output folder for this
        generation : it will be the base path
67     *         against which all file block URLs will be resolved.
68     * @param arguments
69     *         If the template which will be called requires more
        than one argument taken from the model,
70     *         pass them here.
71     * @throws IOException
72     *         This can be thrown in three scenarios : the module
        cannot be found, it cannot be loaded, or
73     *         the model cannot be loaded.
74     * @generated
75     */
76     public Generate(URI modelURI, File targetFolder,
77         List<? extends Object> arguments) throws IOException {
78         initialize(modelURI, targetFolder, arguments);
79     }
80     /**
81     * This allows clients to instantiates a generator with all
        required information.
82     *
83     * @param model
84     *         We'll iterate over the content of this element to
        find Objects matching the first parameter
85     *         of the template we need to call.
86     * @param targetFolder
87     *         This will be used as the output folder for this
        generation : it will be the base path
88     *         against which all file block URLs will be resolved.
89     * @param arguments
90     *         If the template which will be called requires more
        than one argument taken from the model,
91     *         pass them here.
92     * @throws IOException

```

```

93      *           This can be thrown in two scenarios : the module
cannot be found, or it cannot be loaded.
94      * @generated
95      */
96      public Generate(EObject model, File targetFolder,
97          List<? extends Object> arguments) throws IOException {
98          initialize(model, targetFolder, arguments);
99      }
100     /**
101     * This can be used to launch the generation from a standalone
application.
102     *
103     * @param args
104     *           Arguments of the generation.
105     * @generated
106     */
107     public static void main(String[] args) {
108         try {
109             if (args.length < 2) {
110                 System.out.println("Arguments not valid : {model,
111 folder}");
112             } else {
113                 URI modelURI = URI.createFileURI(args[0]);
114                 File folder = new File(args[1]);
115
116                 List<String> arguments = new ArrayList<String>();
117
118                 /*
119                  * If you want to change the content of this method,
do NOT forget to change the "@generated"
120                  * tag in the Javadoc of this method to "@generated
NOT". Without this new tag, any compilation
121                  * of the Acceleo module with the main template that
has caused the creation of this class will
122                  * revert your modifications.
123                  */
124
125                 /*
126                  * Add in this list all the arguments used by the
starting point of the generation
127                  * If your main template is called on an element of
your model and a String, you can
128                  * add in "arguments" this "String" attribute.
129                  */
130
131                 Generate generator = new Generate(modelURI, folder,
arguments);

```

```

131
132         /*
133             * Add the properties from the launch arguments.
134             * If you want to programmatically add new properties
135             , add them in "propertiesFiles"
136             * You can add the absolute path of a properties
137             files, or even a project relative path.
138             * If you want to add another "protocol" for your
139             properties files, please override
140             * "getPropertiesLoaderService(AccelleoService)" in
141             order to return a new property loader.
142             * The behavior of the properties loader service is
143             explained in the Accelleo documentation
144             * (Help -> Help Contents).
145             */
146
147         for (int i = 2; i < args.length; i++) {
148             generator.addPropertyFile(args[i]);
149         }
150
151         generator.doGenerate(new BasicMonitor());
152     }
153     } catch (IOException e) {
154         e.printStackTrace();
155     }
156 }
157
158 /**
159  * Launches the generation described by this instance.
160  *
161  * @param monitor
162  *      This will be used to display progress information
163  *      to the user.
164  * @throws IOException
165  *      This will be thrown if any of the output files
166  *      cannot be saved to disk.
167  * @generated
168  */
169 @Override
170 public void doGenerate(Monitor monitor) throws IOException {
171     /*
172         * TODO if you wish to change the generation as a whole,
173         override this. The default behavior should
174         * be sufficient in most cases. If you want to change the
175         content of this method, do NOT forget to
176         * change the "@generated" tag in the Javadoc of this method
177         to "@generated NOT". Without this new tag,
178         * any compilation of the Accelleo module with the main

```

```

template that has caused the creation of this
168      * class will revert your modifications. If you encounter a
problem with an unresolved proxy during the
169      * generation, you can remove the comments in the following
instructions to check for problems. Please
170      * note that those instructions may have a significant impact
on the performances.
171      */
172
173      //org.eclipse.emf.ecore.util.EcoreUtil.resolveAll(model);
174
175      /*
176      * If you want to check for potential errors in your models
before the launch of the generation, you
177      * use the code below.
178      */
179
180      //if (model != null && model.eResource() != null) {
181      //      List<org.eclipse.emf.ecore.resource.Resource.Diagnostic
> errors = model.eResource().getErrors();
182      //      for (org.eclipse.emf.ecore.resource.Resource.Diagnostic
diagnostic : errors) {
183      //          System.err.println(diagnostic.toString());
184      //      }
185      //}
186
187      super.doGenerate(monitor);
188  }
189  /**
190   * If this generator needs to listen to text generation events,
listeners can be returned from here.
191   *
192   * @return List of listeners that are to be notified when text is
generated through this launch.
193   * @generated
194   */
195  @Override
196  public List<IAccelerioTextGenerationListener>
getGenerationListeners() {
197      List<IAccelerioTextGenerationListener> listeners = super.
getGenerationListeners();
198      /*
199      * TODO if you need to listen to generation event, add
listeners to the list here. If you want to change
200      * the content of this method, do NOT forget to change the "
@generated" tag in the Javadoc of this method
201      * to "@generated NOT". Without this new tag, any compilation

```

```

    of the Acceleio module with the main template
202     * that has caused the creation of this class will revert
your modifications.
203     */
204     return listeners;
205 }
206 /**
207     * If you need to change the way files are generated, this is
your entry point.
208     * <p>
209     * The default is {@link org.eclipse.acceleio.engine.generation.
strategy.DefaultStrategy}; it generates
210     * files on the fly. If you only need to preview the results,
return a new
211     * {@link org.eclipse.acceleio.engine.generation.strategy.
PreviewStrategy}. Both of these aren't aware of
212     * the running Eclipse and can be used standalone.
213     * </p>
214     * <p>
215     * If you need the file generation to be aware of the workspace (
A typical example is when you wanna
216     * override files that are under clear case or any other VCS that
could forbid the overriding), then
217     * return a new {@link org.eclipse.acceleio.engine.generation.
strategy.WorkspaceAwareStrategy}.
218     * <b>Note</b>, however, that this <b>cannot</b> be used
standalone.
219     * </p>
220     * <p>
221     * All three of these default strategies support merging through
JMerge.
222     * </p>
223     *
224     * @return The generation strategy that is to be used for
generations launched through this launcher.
225     * @generated
226     */
227     @Override
228     public IAcceleioGenerationStrategy getGenerationStrategy() {
229         return super.getGenerationStrategy();
230     }
231     /**
232     * This will be called in order to find and load the module that
will be launched through this launcher.
233     * We expect this name not to contain file extension, and the
module to be located beside the launcher.
234     *

```

```

235     * @return The name of the module that is to be launched.
236     * @generated
237     */
238     @Override
239     public String getModuleName() {
240         return MODULE_FILE_NAME;
241     }
242     /**
243     * If the module(s) called by this launcher require properties
244     * files, return their qualified path from
245     * here. Take note that the first added properties files will take
246     * precedence over subsequent ones if they
247     * contain conflicting keys.
248     *
249     * @return The list of properties file we need to add to the
250     * generation context.
251     * @see java.util.ResourceBundle#getBundle(String)
252     * @generated
253     */
254     @Override
255     public List<String> getProperties() {
256         /*
257         * If you want to change the content of this method, do NOT
258         * forget to change the "@generated"
259         * tag in the Javadoc of this method to "@generated NOT".
260         * Without this new tag, any compilation
261         * of the Acceleo module with the main template that has
262         * caused the creation of this class will
263         * revert your modifications.
264         */
265         /*
266         * TODO if your generation module requires access to
267         * properties files, add their qualified path to the list here.
268         */
269         * Properties files can be located in an Eclipse plug-in or
270         * in the file system (all Acceleo projects are Eclipse
271         * plug-in). In order to use properties files located in an
272         * Eclipse plugin, you need to add the path of the properties
273         * files to the "propertiesFiles" list:
274         *
275         * final String prefix = "platform:/plugin/";
276         * final String pluginName = "org.eclipse.acceleo.module.
277         * sample";
278         * final String packagePath = "/org/eclipse/acceleo/module/
279         * sample/properties/";
280         * final String fileName = "default.properties";

```

```

271         * propertiesFiles.add(prefix + pluginName + packagePath +
        fileName);
272         *
273         * With this mechanism, you can load properties files from
        your plugin or from another plugin.
274         *
275         * You may want to load properties files from the file system
        , for that you need to add the absolute path of the file:
276         *
277         * propertiesFiles.add("C:\Users\MyName\MyFile.properties");
278         *
279         * If you want to let your users add properties files located
        in the same folder as the model:
280         *
281         * if (EMFPlugin.IS_ECLIPSE_RUNNING && model != null && model
        .eResource() != null) {
282             *     propertiesFiles.addAll(AccileoEngineUtils.
        getPropertiesFilesNearModel(model.eResource()));
283             * }
284             *
285             * To learn more about Properties Files, have a look at the
        Accileo documentation (Help -> Help Contents).
286         */
287         return propertiesFiles;
288     }
289     /**
290     * Adds a properties file in the list of properties files.
291     *
292     * @param propertiesFile
293     *         The properties file to add.
294     * @generated
295     * @since 3.1
296     */
297     @Override
298     public void addPropertiesFile(String propertiesFile) {
299         this.propertiesFiles.add(propertiesFile);
300     }
301     /**
302     * This will be used to get the list of templates that are to be
        launched by this launcher.
303     *
304     * @return The list of templates to call on the module {@link #
        getModuleName()}.
305     * @generated
306     */
307     @Override
308     public String[] getTemplateName() {

```



```

309         return TEMPLATE_NAMES;
310     }
311     /**
312      * This can be used to update the resource set's package registry
313      * with all needed EPackages.
314      *
315      * @param resourceSet
316      *         The resource set which registry has to be updated.
317      * @generated
318      */
319     @Override
320     public void registerPackages(ResourceSet resourceSet) {
321         super.registerPackages(resourceSet);
322         if (!isInWorkspace(org.eclipse.papyrus.sysml.SysmlPackage.
class)) {
323             resourceSet.getPackageRegistry().put(org.eclipse.papyrus.
sysml.SysmlPackage.eINSTANCE.getNsURI(), org.eclipse.papyrus.sysml.
SysmlPackage.eINSTANCE);
324         }
325         if (!isInWorkspace(org.eclipse.papyrus.sysml.modelelements.
ModelelementsPackage.class)) {
326             resourceSet.getPackageRegistry().put(org.eclipse.papyrus.
sysml.modelelements.ModelelementsPackage.eINSTANCE.getNsURI(), org.
eclipse.papyrus.sysml.modelelements.ModelelementsPackage.
eINSTANCE);
327         }
328         if (!isInWorkspace(org.eclipse.papyrus.sysml.blocks.
BlocksPackage.class)) {
329             resourceSet.getPackageRegistry().put(org.eclipse.papyrus.
sysml.blocks.BlocksPackage.eINSTANCE.getNsURI(), org.eclipse.
papyrus.sysml.blocks.BlocksPackage.eINSTANCE);
330         }
331         if (!isInWorkspace(org.eclipse.papyrus.sysml.portandflows.
PortandflowsPackage.class)) {
332             resourceSet.getPackageRegistry().put(org.eclipse.papyrus.
sysml.portandflows.PortandflowsPackage.eINSTANCE.getNsURI(), org.
eclipse.papyrus.sysml.portandflows.PortandflowsPackage.eINSTANCE);
333         }
334         if (!isInWorkspace(org.eclipse.papyrus.sysml.constraints.
ConstraintsPackage.class)) {
335             resourceSet.getPackageRegistry().put(org.eclipse.papyrus.
sysml.constraints.ConstraintsPackage.eINSTANCE.getNsURI(), org.
eclipse.papyrus.sysml.constraints.ConstraintsPackage.eINSTANCE);
336         }
337         if (!isInWorkspace(org.eclipse.papyrus.sysml.activities.
ActivitiesPackage.class)) {
338             resourceSet.getPackageRegistry().put(org.eclipse.papyrus.

```

```

    sysml.activities.ActivitiesPackage.eINSTANCE.getNsURI(), org.
    eclipse.papyrus.sysml.activities.ActivitiesPackage.eINSTANCE);
338     }
339     if (!isInWorkspace(org.eclipse.papyrus.sysml.allocations.
    AllocationsPackage.class)) {
340         resourceSet.getPackageRegistry().put(org.eclipse.papyrus.
    sysml.allocations.AllocationsPackage.eINSTANCE.getNsURI(), org.
    eclipse.papyrus.sysml.allocations.AllocationsPackage.eINSTANCE);
341     }
342     if (!isInWorkspace(org.eclipse.papyrus.sysml.requirements.
    RequirementsPackage.class)) {
343         resourceSet.getPackageRegistry().put(org.eclipse.papyrus.
    sysml.requirements.RequirementsPackage.eINSTANCE.getNsURI(), org.
    eclipse.papyrus.sysml.requirements.RequirementsPackage.eINSTANCE);
344     }
345     if (!isInWorkspace(org.eclipse.papyrus.sysml.interactions.
    InteractionsPackage.class)) {
346         resourceSet.getPackageRegistry().put(org.eclipse.papyrus.
    sysml.interactions.InteractionsPackage.eINSTANCE.getNsURI(), org.
    eclipse.papyrus.sysml.interactions.InteractionsPackage.eINSTANCE);
347     }
348     if (!isInWorkspace(org.eclipse.papyrus.sysml.statemachines.
    StatemachinesPackage.class)) {
349         resourceSet.getPackageRegistry().put(org.eclipse.papyrus.
    sysml.statemachines.StatemachinesPackage.eINSTANCE.getNsURI(), org.
    eclipse.papyrus.sysml.statemachines.StatemachinesPackage.
    eINSTANCE);
350     }
351     if (!isInWorkspace(org.eclipse.papyrus.sysml.usecases.
    UsecasesPackage.class)) {
352         resourceSet.getPackageRegistry().put(org.eclipse.papyrus.
    sysml.usecases.UsecasesPackage.eINSTANCE.getNsURI(), org.eclipse.
    papyrus.sysml.usecases.UsecasesPackage.eINSTANCE);
353     }
354     if (!isInWorkspace(org.eclipse.papyrus.sysml.blocks.
    BlocksPackage.class)) {
355         resourceSet.getPackageRegistry().put(org.eclipse.papyrus.
    sysml.blocks.BlocksPackage.eINSTANCE.getNsURI(), org.eclipse.
    papyrus.sysml.blocks.BlocksPackage.eINSTANCE);
356     }
357     if (!isInWorkspace(org.eclipse.uml2.uml.UMLPackage.class)) {
358         resourceSet.getPackageRegistry().put(org.eclipse.uml2.uml.
    UMLPackage.eINSTANCE.getNsURI(), org.eclipse.uml2.uml.UMLPackage.
    eINSTANCE);
359     }
360     /*
361     * If you want to change the content of this method, do NOT

```

```

forget to change the "@generated"
362      * tag in the Javadoc of this method to "@generated NOT".
Without this new tag, any compilation
363      * of the Acceleo module with the main template that has
caused the creation of this class will
364      * revert your modifications.
365      */
366
367    /*
368      * If you need additional package registrations, you can
register them here. The following line
369      * (in comment) is an example of the package registration for
UML.
370      *
371      * You can use the method "isInWorkspace(Class c)" to check
if the package that you are about to
372      * register is in the workspace.
373      *
374      * To register a package properly, please follow the
following conventions:
375      *
376      * If the package is located in another plug-in, already
installed in Eclipse. The following content should
377      * have been generated at the beginning of this method. Do
not register the package using this mechanism if
378      * the metamodel is located in the workspace.
379      *
380      * if (!isInWorkspace(UMLPackage.class)) {
381      *     // The normal package registration if your metamodel
is in a plugin.
382      *     resourceSet.getPackageRegistry().put(UMLPackage.
eNS_URI, UMLPackage.eINSTANCE);
383      * }
384      *
385      * If the package is located in another project in your
workspace, the plugin containing the package has not
386      * been register by EMF and Acceleo should register it
automatically. If you want to use the generator in
387      * stand alone, the regular registration (seen a couple lines
before) is needed.
388      *
389      * To learn more about Package Registration, have a look at
the Acceleo documentation (Help -> Help Contents).
390      */
391    }
392    /**
393      * This can be used to update the resource set's resource factory

```

```

        registry with all needed factories.
394     *
395     * @param resourceSet
396     *         The resource set which registry has to be updated.
397     * @generated
398     */
399     @Override
400     public void registerResourceFactories(ResourceSet resourceSet) {
401         super.registerResourceFactories(resourceSet);
402         /*
403          * If you want to change the content of this method, do NOT
404          * forget to change the "@generated"
405          * tag in the Javadoc of this method to "@generated NOT".
406          * Without this new tag, any compilation
407          * of the Acceleo module with the main template that has
408          * caused the creation of this class will
409          * revert your modifications.
410          */
411          * TODO If you need additional resource factories
412          * registrations, you can register them here. the following line
413          * (in comment) is an example of the resource factory
414          * registration for UML.
415          *
416          * If you want to use the generator in stand alone, the
417          * resource factory registration will be required.
418          *
419          * To learn more about the registration of Resource Factories
420          * , have a look at the Acceleo documentation (Help -> Help Contents)
421          * .
422          */
423          // resourceSet.getResourceFactoryRegistry().
424          getExtensionToFactoryMap().put(UMLResource.FILE_EXTENSION,
425          UMLResource.Factory.INSTANCE);
426      }
427  }

```

Código-fonte 7: generate.mtl

```

1 [comment encoding = UTF-8 /]
2 [**
3  * The documentation of the module generate.
4  */]
5 [module generate('http://www.eclipse.org/papyrus/0.7.0/SysML', 'http
: //www.eclipse.org/papyrus/0.7.0/SysML/Blocks', 'http://www.

```

```

        eclipse.org/uml2/5.0.0/UML'))]
6
7 [import larissamodel2simulation::services::umlservices /]
8
9 /**
10  * The documentation of the template generateElement.
11  * @param aModel
12  */]
13 [template public generateElement(m : Model)]
14 [comment @main/]
15 [file ('uavconf.ini', false, 'UTF-8')]
16 [for (p:Package | m.eAllContents(Package))]
17     [getPackage( p,'UAV')/]
18 [/for]
19 [/file]
20 [/template]
21
22 [template public getPackage(p:Package, n:String)]
23     [for (c:NamedElement | p.eAllContents(NamedElement))]
24         [generateClassDeclararions(c)/]
25     [/for]
26 [/template]
27
28 [template public generateClassDeclararions(c:Class) ]
29 [for (p: Property | c.attribute) ]
30     private [p.name/] [p.default/];
31 [/for]
32
33 [for (es: Stereotype | c.getAppliedStereotypes()) ]
34     [if (es.name = 'RadioModem' )]
35     #####
36     #           Parameters for the ConnectionManager           #
37     #####
38     [for ( pn: Property | es.allAttributes()) ]
39         [if (pn.name = 'standard')]
40             [if (getTaggedValue(c, es, pn.name).toString().first(9) = '
ieee802154')]
41                 [for ( ps: Property | es.allAttributes()) ]
42                     [if (ps.name = 'frequency')]
43                         [if (getTaggedValue(c, es, ps.name) = '_433')]
44 **.connectionManager.carrierFrequency = 433Hz # - Hz
45                     [/if]
46                         [if (getTaggedValue(c, es, ps.name) = '_868')]
47 **.connectionManager.carrierFrequency = 868Hz # - Hz
48                     [/if]
49                         [if (getTaggedValue(c, es, ps.name) = '_915')]
50 **.connectionManager.carrierFrequency = 915Hz # - Hz

```

```

51             [/if]
52             [if (getTaggedValue(c, es, ps.name) = '_2400')]
53 **.connectionManager.carrierFrequency = 2.4e9Hz # - Hz
54             [/if]
55             [if (getTaggedValue(c, es, ps.name) = '_5000')]
56 **.connectionManager.carrierFrequency = 5.0e9Hz # - Hz
57             [/if]
58         [/if]
59         [if (ps.name = 'rfPower')]
60 # max transmission power - mW
61 **.connectionManager.pMax = [getTaggedValue(c, es, ps.name)]mW
62         [/if]
63         [if (ps.name = 'sensitivity')]
64 # signal attenuation threshold - dBm
65 **.connectionManager.sat = [getTaggedValue(c, es, ps.name)]dBm
66         [/if]
67     [/for]
68 # path loss coefficient alpha
69 **.connectionManager.alpha = 2.5
70 **.connectionManager.sendDirect = false
71     [/if]
72 [/if]
73 [/for]
74
75 #####
76 #           Parameters for the Host           #
77 #####
78 ##### Phy parameters #####
79 [for ( pn: Property | es.allAttributes() ) ]
80     [if (pn.name = 'standard')]
81         [if (getTaggedValue(c, es, pn.name).toString().first(9) = '
            ieee802154')]
82 **.phy.usePropagationDelay = false
83 **.phy.useThermalNoise = true
84 **.phy.analogueModels = xmldoc("config.xml")
85         [for ( ps: Property | es.allAttributes() ) ]
86             [if (ps.name = 'rfPower')]
87 **.phy.maxTXPower = [getTaggedValue(c, es, ps.name)]mW
88             [/if]
89             [if (ps.name = 'sensitivity')]
90 **.phy.sensitivity = [getTaggedValue(c, es, ps.name)]dBm
91             [/if]
92         [/for]
93     [/if]
94     [if (getTaggedValue(c, es, pn.name).toString().first(9) = '
        ieee80211')]
95         [for ( ps: Property | es.allAttributes() ) ]

```

```

96             [if (ps.name = 'frequency')]
97                 [if (getTaggedValue(c, es, ps.name) = '_433')]
98 **.radio.carrierFrequency = 433Hz # - Hz
99                 [/if]
100            [if (getTaggedValue(c, es, ps.name) = '_868')]
101 **.radio.carrierFrequency = 868Hz # - Hz
102            [/if]
103            [if (getTaggedValue(c, es, ps.name) = '_915')]
104 **.radio.carrierFrequency = 915Hz # - Hz
105            [/if]
106            [if (getTaggedValue(c, es, ps.name) = '_2400')]
107 **.radio.carrierFrequency = 2.4e9Hz # - Hz
108            [/if]
109            [if (getTaggedValue(c, es, ps.name) = '_5000')]
110 **.radio.carrierFrequency = 5.0e9Hz # - Hz
111            [/if]
112        [/if]
113        [if (ps.name = 'rfPower')]
114 # max transmission power - mW
115 **.radio.transmitterPower = [getTaggedValue(c, es, ps.name)]mW
116        [/if]
117        [if (ps.name = 'sensitivity')]
118 # signal attenuation threshold - dBm
119 **.radio.sensitivity = [getTaggedValue(c, es, ps.name)]dBm
120        [/if]
121    [/for]
122 [/if]
123 [/if]
124 [/for]
125 ##### Phy parameters #####
126 [/if]
127 [if (es.name = 'Battery' )]
128 #####
129 #         Parameters for the Battery #
130 #####
131     [for ( ps: Property | es.allAttributes() ) ]
132         [if (ps.name = 'voltage')]
133 **.battery.voltage = [getTaggedValue(c, es, ps.name)]
134         [/if]
135         [if (ps.name = 'ampere')]
136 **.battery.nominal = [getTaggedValue(c, es, ps.name)]
137 **.battery.capacity = [getTaggedValue(c, es, ps.name)]
138         [/if]
139     [/for]
140 ##### Battery parameters #####
141 [/if]
142 [/for]

```

```

143
144
145 [/template]
146
147 [template public generateClassDeclararions(e:NamedElement) ? (e.
    hasStereotype('Block'))]
148 [file (e.name + '.txt', false, 'UTF-8')]
149
150 /*
151 O nome do elemento eh [e.name/] de [e.Container().eGet('name')/]
    Estereotype: [e.getAppliedStereotypes()/]
152 */
153
154 [for (es: Stereotype | e.getAppliedStereotypes()) ]
155     [for ( ps: Property | es.attribute) ]
156         private tipo: [ps.type.name/] nome: [ps.name/] valor: ;
157     [/for]
158     [for ( el: Element | es.attribute) ]
159         Estereotype: [es.name/]
160         Element: [el.eGet('name')/]
161         [if (el.hasStereotype(el,es.name))]
162             Valor: [el.eGet('name')/] -> [getTaggedValuet(el, es, el.eGet
                ('name'))/]
163         [/if]
164         Valor => [getTaggedValuet(el, es, el.eGet('name'))/]
165     [/for]
166 [/for]
167
168
169 [/file]
170 [/template]
171
172 [template public generateClassDeclararions(b:Block)]
173 [file (b.base_Class.name + '.blk', false, 'UTF-8')]
174
175 /*
176 O nome do modelo eh [b.base_Class.name/] Estereotype Classe: [b.
    base_Class.getAppliedStereotypes().eGet('name')/]
177 Escreva um comentario.
178 */
179
180 class [b.base_Class.name/] {
181 [for (p: Property | b.base_Class.attribute) ]
182     private [p.type.name/] [p.name/];
183 [/for]
184 [for (es: Stereotype | b.base_Class.getAppliedStereotypes()) ]
185     [for ( ps: Property | es.attribute) ]

```

```

186     private tipo: [ps.type.name/] nome: [ps.name/] valor: ;
187 [/for]
188 [for ( el: Element | es.attribute) ]
189     Estereotype: [es.name/]
190     Element: [el.eGet('name')/]
191     [if (el.hasStereotype(el,es.name))]
192         Valor: [el.eGet('name')/] -> [getTaggedValuet(el, es, el.eGet
        ('name'))/]
193     [/if]
194     Valor => [getTaggedValuet(el, es, el.eGet('name'))/]
195 [/for]
196 [/for]
197 }
198 [/file]
199 [/template]

```

Código-fonte 8: UMLservices.java

```

1 package larissamodel2simulation.services;
2
3 import java.util.ArrayList;
4 import java.util.Collection;
5 import java.util.List;
6
7 import org.eclipse.uml2.uml.Element;
8 import org.eclipse.uml2.uml EnumerationLiteral;
9 import org.eclipse.uml2.uml.Stereotype;
10
11 public class UMLservices {
12     public boolean hasStereotype(Element element, String stereotypeName
        ) {
13         return element.getAppliedStereotype(stereotypeName) != null;
14     }
15
16     public List<Object> getTaggedValue(Element element, String
        stereotypeName, String property) {
17         Stereotype stereotype = element.getAppliedStereotype(
        stereotypeName);
18         List<Object> result = new ArrayList<Object>();
19         if (stereotype == null) {
20             System.out.println("Nao Achou");
21             return result;
22         }
23
24         Object value = element.getValue(stereotype, property);
25         if (value instanceof Collection) {
26             result.addAll((Collection)value);

```

```

27     } else {
28         result.add(value);
29     }
30     return result;
31 }
32
33 public Object getTaggedValue(Element element, Stereotype stereotype
    , String property) {
34     String result = new String();
35     if (stereotype == null) {
36         return result;
37     }
38     Object value = element.getValue(stereotype, property);
39     if (value instanceof EnumerationLiteral) {
40         EnumerationLiteral literal=(EnumerationLiteral)value;
41 //         System.out.println("Eh Enumeration: " + property);
42 //         return (literal.getInstance() != null) ? literal.
        getInstance() : literal;
43         return literal.getName();
44     } else {
45 //         System.out.println("Nao eh Enumeration: " + property);
46         result = value.toString();
47     }
48     return result;
49 }
50
51 public String getTaggedValuet(Element element, Stereotype
    stereotype, String property) {
52     String result = new String();
53     result = stereotype.getName().concat(property);
54     return result;
55 }
56 }

```

Código-fonte 9: umlservices.mtl

```

1 [comment encoding = Cp1252 /]
2 [module umlservices('http://www.eclipse.org/uml2/5.0.0/UML', 'http://
    www.eclipse.org/papyrus/0.7.0/SysML', 'http://www.eclipse.org/
    papyrus/0.7.0/SysML/Blocks' )/]
3
4 [query public hasStereotype(arg0 : Element, arg1 : String) : Boolean
5     = invoke('larissamodel2simulation.services.UMLservices', '
        hasStereotype(org.eclipse.uml2.uml.Element, java.lang.String)',
        Sequence{arg0, arg1})]
6 [/]
7

```

```

8 [query public getTaggedValue(arg0 : Element, arg1 : String, arg2 :
   String) : Sequence(OclAny)
9   = invoke('larissamodel2simulation.services.UMLservices', '
      getTaggedValue(org.eclipse.uml2.uml.Element, java.lang.String,
      java.lang.String)', Sequence{arg0, arg1, arg2})
10 /]
11
12 [query public getTaggedValue(arg0 : Element, arg1 : Stereotype, arg2
   : String) : OclAny
13   = invoke('larissamodel2simulation.services.UMLservices', '
      getTaggedValue(org.eclipse.uml2.uml.Element, org.eclipse.uml2.uml.
      Stereotype, java.lang.String)', Sequence{arg0, arg1, arg2})
14 /]
15
16 [query public getTaggedValuet(arg0 : Element, arg1 : Stereotype, arg2
   : String) : String
17   = invoke('larissamodel2simulation.services.UMLservices', '
      getTaggedValuet(org.eclipse.uml2.uml.Element, org.eclipse.uml2.uml.
      Stereotype, java.lang.String)', Sequence{arg0, arg1, arg2})
18 /]

```

Os demais códigos que compõem o *larissamodel2siluation* estão disponíveis no site do LSEC¹.

¹ <<http://www.lsec.icmc.usp.br/larissa/>>