Desenvolvimento de um mecanismo plug and play para o arranjo inteligente de sensores em sistemas aéreos não tripulados

Rayner de Melo Pires

Desenvolvimento de um mecanismo plug and play para o arranjo inteligente de sensores em sistemas aéreos não tripulados

Rayner de Melo Pires

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

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

USP – São Carlos
Maio de 2014

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

Pires, Rayner de Melo
P667d Desenvolvimento de um mecanismo plug and play para o arranjo inteligente de sensores em sistemas aéreos não tripulados / Rayner de Melo Pires; orientadora Kalinka Regina Lucas Jaquie Castelo Branco. -- São Carlos, 2014.
82 p.

Dissertação (Mestrado - Programa de Pós-Graduação em Ciências de Computação e Matemática Computacional) -- Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, 2014.

1. aeronaves não tripuladas. 2. projeto de protocolos. 3. verificação de modelos. 4. arranjos inteligentes de sensores. I. Branco, Kalinka Regina Lucas Jaquie Castelo, orient. II. Título.

(. . .) A educação deve, então, ser não apenas uma transmissão de cultura, mas também um fornecedor de visões alternativas do mundo e um fortalecedor da vontade de explorá-las.

Jerome S. Bruner

Dedico este trabalho à minha família:
Fábio Pires, Jussara Rosa, Taynara, Plínio e Fábio.

Agradecimentos

Agradeço a todos que me acompanharam e me ajudaram neste trabalho, principalmente os amigos de laboratório Adimara Colturato, Adriano Belfort, Arthur Chaves, Daniel Pigatto, Emerson Marconato, Jéssica Vida, Natássya Barlate e Paulo Gurgel.

Em especial, sou muito grato à professora Kalinka Castelo Branco pelo seu entusiasmo contagiante, pelo conhecimento repassado e por ter me orientado tão bem desde o começo. Todos vocês contribuiram muito para o meu crescimento pessoal e profissional.

Agradeço também ao ICMC/USP pela excelente infraestrutura oferecida, à CAPES pela bolsa disponibilizada no início deste curso, e à FAPESP pelo financiamento do projeto 2011/044161, sem os quais não teria sido possível realizar este trabalho do começo ao fim.

Resumo

O uso de aeronaves não tripuladas (VANTs) tem crescido substancialmente nos últimos anos, tanto no campo militar quanto no civil. *Roadmaps* preveem que em um futuro próximo essas aeronaves compartilhem o espaço aéreo com aeronaves convencionais, exigindo novas arquiteturas de sistemas embarcados que possam garantir uma operação coordenada e segura desses robôs. A maior parte das suas missões baseia-se fortemente em um conjunto de sensores transportados pela aeronave como parte da *payload* da missão. Contudo, não é trivial a integração de diferentes missões em diferentes aeronaves, visto que ainda não há uma padronização para a comunicação nesses robôs. Para possibilitar essa associação foi proposto neste trabalho a criação de um *middleware*. Para que se pudesse entender sobre a área de conhecimento dos VANTs realizou-se uma pesquisa sobre esses veículos e suas aplicações e então um protocolo chamado *Smart Sensor Protocol* (SSP) foi modelado, utilizando-se de técnicas formais para isso. O comportamento do protocolo está modelado com diagrama de estados, seguindo uma gramática escrita utilizando a forma BNF. Este modelo foi verificado com a ferramenta UPPAAL e sua implementação testada em placas Arduino. Os resultados dos testes mostraram que o modelo é viável para o ambiente de embarcados críticos visto que ele provê as funcionalidades necessárias neste cenário sem acrescentar um *overhead* na comunicação.

Abstract

UNMANNED Aerial Vehicles applications have grown substantially in recent years, both in military and civil fields. Roadmaps predict that in the near future these aircrafts will share the airspace with the conventional planes, requiring new architectures for embedded systems which may ensure a coordinated and safe operation. Most of its tasks are mainly based on a set of sensors carried by the aircraft as part of its payload. However, it is not trivial to integrate different missions in different aircraft plataforms, since there is no standardization for communication in such robots yet. To enable this type of association it was proposed in this master's project the designing of a middleware. It has been carried out a bibliographic review to find out the state-of-the-art in such field, including the specific applications in UAVs, and then a protocol has been modeled following formal techniques. This protocol is called Smart Sensor Protocol (SSP). The SSP's behavior was modeled through state diagrams according to a grammar described using BNF form. This model was verified with the UPPAAL tool and its implementation was run and tested on Arduino boards. The test results pointed out that the model is feasible for critical embedded environments since it provides the necessary functionality in this scenario without addition of an overhead in its communication.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Siglas

BNF	*Backus-Naur Form*
FAA	*U.S. Federal Aviation Administration*
FDTs	*Formal Description Techniques*
IED	*Improvised Explosive Device*
MOSA	*Mission Oriented Sensors Array*
OSD	*U.S. Office of the Secretary of Defense*
SANT	Sistema de Aeronave Não Tripulada
SSP	*Smart Sensor Protocol*
UAS	*Unmanned Aircraft System*
UAV	*Unmanned Aerial Vehicle*
UGV	*Unmanned Ground Vehicle*
UMS	*Unmanned Maritime System*
USV	*Unmanned Surface Vehicle*
UUSV	*Unmanned Undersea Vehicle*
VANT	Veículo Aéreo Não Tripulado

CAPÍTULO

1

Introdução

Os sistemas aéreos não tripulados são altamente cobiçados nas forças armadas contemporâneas pela sua versatilidade e persistência. Ao executar tarefas como vigilância, espionagem de sinais designação precisa de alvos, detecção de minas e reconhecimento químico, biológico, radiológico e nuclear esses sistemas deram contribuições fundamentais para a iniciativa norte americana contra o terrorismo mundial. Foi mensurado em (Clapper Jr et al., 2009) que uma coalizão de sistemas aéreos não tripulados lançados à mão havia voado quase 500.000 horas de voo em apoio a missões militares no Iraque, juntamente com outros sistemas não tripulados, como UGVs (*Unmanned Ground Vehicles*), UMSs (*Unmanned Maritime Systems*) e IEDs (*Improvised Explosive Devices*).

No *roadmap* [1] mais recente da área, publicado pelo OSD (*U.S. Office of the Secretary of Defense*), Clapper Jr et al. apontam também uma perspectiva de evolução para atributos de desempenho que sistemas não tripulados devem apresentar. Logo em primeiro lugar, citam a progressão do nível de autonomia desses sistemas, evoluindo do alto nível de controle e intervenção humanos atual para um comportamento tático completamente autônomo até o ano 2034. Outra característica essencial desejável será equipamentos de missão que possam ser intercambiados entre plataformas e, potencialmente, até mesmo entre domínios (UAVs $\longleftrightarrow$ UGVs $\longleftrightarrow$ UMVs $\longleftrightarrow$ UAVs). Hoje, a maioria das *payloads* [2] são projetadas para integração com uma única plataforma. Ao fornecer permutabilidade entre plataformas e domínios, esses sistemas se tornarão mais flexíveis no fator variabilidade para a realização de missões específicas em determinados tipos de circunstâncias.

[1] Um conjunto de orientações, instruções, planos ou explicações.

[2] Carga útil da aeronave, que na maioria das vezes são sensores, podendo ser também armamento ou outros.

Com contínuas adaptações evolutivas desses sistemas ao longo de 33 anos – período estimado por *roadmaps* encontrados na literatura (Aldridge Jr e Stenbit, 2002; Cambone et al., 2005; Clapper Jr et al., 2007, 2009) –, órgãos de controle e regulamentação têm sido incentivados a estudar e criar, juntamente com congressistas, leis que regimentarão o uso desses sistemas não tripulados em outros ambientes que não o militar. A regulamentação do espaço aéreo, por exemplo, para uso por UAVs estimula o aumento da pesquisa e da utilização desses veículos em missões que tenham caráter civil.

1.1 Motivação

O crescimento da utilização de VANTs (Veículos Aéreos Não Tripulados) tanto em ambiente militar quanto em ambiente civil, onde são mais conhecidos como Drones, tem feito com que eles se tornem cada vez mais comuns, ocasionando sua comercialização de forma mais ampla (Aldridge Jr e Stenbit, 2002; Trindade Jr et al., 2002; Cambone et al., 2005; Clapper Jr et al., 2009). A grande variedade de aplicações para esses robôs deixam bastante heterogêneas a combinação e a programação de dispositivos de sensoriamento nessas aeronaves.

Nesse contexto, viu-se a oportunidade de criar um mecanismo de inteligência para o acoplamento desses sensores com os VANTs e para a execução de missões por esses veículos. Com esse mecanismo, o processamento da missão fica desassociado do sistema de controle da aeronave, ficando sob responsabilidade exclusiva do módulo independente de processamento de missão, pertencente ao MOSA (*Mission Oriented Sensors Array*) (Pires et al., 2011), deixando o VANT com o encargo apenas de transportar esse módulo durante a execução da missão.

1.2 Objetivos

O objetivo deste trabalho de mestrado é o desenvolvimento de um mecanismo *plug and play* para acoplamento entre uma *payload* e o restante de um sistema de aeronave não tripulada, como representado na Figura 1.1. Esse mecanismo é representado pelo protocolo desenvolvido, nomeado SSP (*Smart Sensor Protocol*).

Ao ser acoplada fisicamente à aeronave, a *payload* (considerada aqui como sendo uma unidade MOSA) iniciará um procedimento *plug and play* que provê a configuração automática das partes, ou ainda, a autoconfiguração da aeronave e da unidade MOSA de tal forma que elas possam colaborar para o cumprimento da missão definida e a ser executada no voo. O objetivo final desse mecanismo de acoplamento é contribuir para o progresso da autonomia nessas aeronaves, um esforço impulsionado por (Clapper Jr et al., 2009).

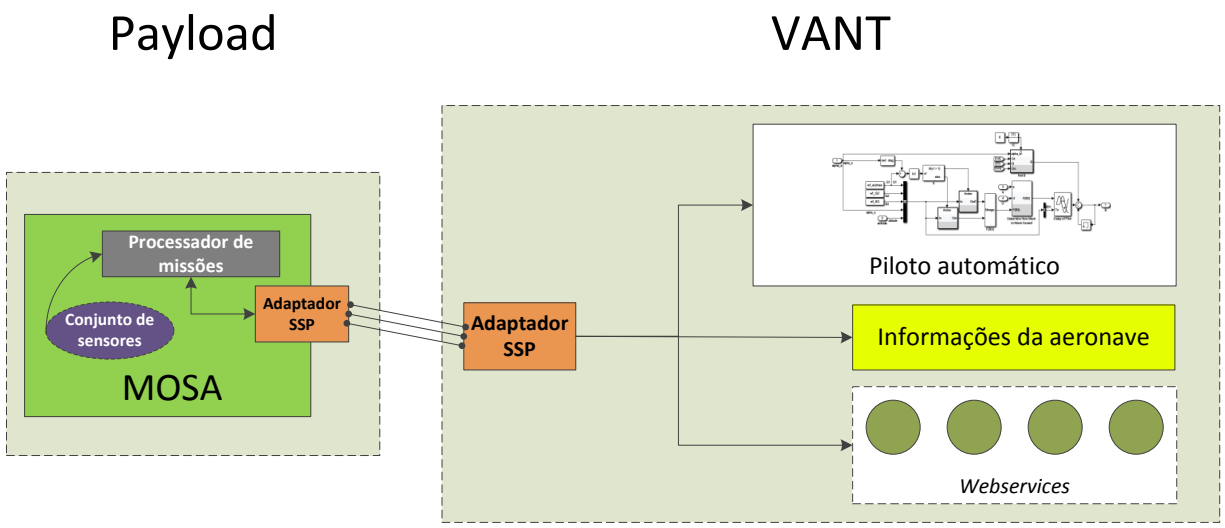


Figura 1.1: Ilustração do mecanismo de interfaceamento provido pelo protocolo SSP

E, por fim, também faz parte dos objetivos deste trabalho a validação do sistema desenvolvido quanto a aspectos funcionais e não funcionais, tais como a verificação por realização de progresso, ausência de *deadlocks* e terminação.

1.3 Organização do texto

Este texto está estruturado da seguinte maneira: O Capítulo 2 apresenta os trabalhos das áreas relacionadas com o que é proposto nesta dissertação. Começa apresentando o cenário de pesquisa sobre a integração entre plataformas heterogêneas, depois apresenta as missões que têm sido projetadas para os VANTs e, por último, apresenta os trabalhos relacionados com o projeto de protocolos *plug-and-play*.

O Capítulo 3 inicia-se apresentando os conceitos de aeronave não tripulada e de sistema aéreo não tripulado, e algumas classificações pertinentes a eles, um modelo de referência para esses sistemas que permite a inserção deste trabalho neste campo, e por fim, algumas aplicações desses sistemas. Em seguida entra na área de projetos de protocolos, dissertando sobre as fases de especificação, validação e teste de modelos.

O Capítulo 4 apresenta o projeto do protocolo SSP: a sua arquitetura, a estrutura das mensagens e o modelo da comunicação. Em seguida apresenta a gramática do protocolo, bem como sua sintaxe e semântica.

O Capítulo 5 primeiramente aborda os experimentos realizados com a ferramenta UPPAAL (UPPAAL Team, 2013): a modelagem do sistema como um todo, a simulação de sua execução e depois as regras criadas para verificar as propriedades do protocolo. Posteriormente, o capítulo apresenta o experimento realizado com o protocolo implementado em duas placas Arduino Mega 2560, utilizadas como *middleware* SSP.

Por último, o Capítulo 6 discorre sobre os resultados obtidos com este trabalho e suas contribuições para o campo. Ainda, apresenta também a produção intelectual do mestrando durante o curso e uma seção de sugestões de trabalhos futuros, que podem ser derivados do trabalho aqui apresentado.

CAPÍTULO

2

Trabalhos Relacionados

2.1 Integração entre múltiplos sensores e múltiplas plataformas não tripuladas

O campo de pesquisa sobre VANTs e suas aplicações é bastante amplo. A literatura que aborda estes veículos é vasta e é encontrada em diversas áreas de pesquisa (Wang et al., 2007; Colomina et al., 2008; Koski et al., 2009; Hodgson et al., 2010; Gaspar et al., 2011; Jorge et al., 2011; Zhang e Kovacs, 2012; Shariff et al., 2012). Contudo, em busca de trabalhos relacionados ao processamento de missões por sistemas não tripulados, os textos encontrados (Tso et al., 1999; Ferrell e Ferrell, 2001; Jacques, 2002; Drury et al., 2006; Merino et al., 2006; Aldridge Jr e Stenbit, 2002; Cambone et al., 2005; Clapper Jr et al., 2007, 2009; Valavanis, 2007; Cooper e Goodrich, 2008; Donley e Schwartz, 2009; Pitre et al., 2009; Karaman et al., 2009; Maza et al., 2010; George et al., 2011; van Willigen et al., 2011) não evidenciaram uma sistematização do processo de autonomia de decisão nas implementações apresentadas. A padronização entre arquiteturas ou modelos de comunicação e mecanismos *plug and play* para conexão, comunicação e cooperação para processamento deste tipo de tarefas são características aqui propostas e que não se encontram presentes nos trabalhos supracitados.

Cada missão desenvolvida para um veículo remotamente pilotado exige todo ou quase todo o processo de implantação dos subsistemas na aeronave, que envolve as tarefas de (i) elaboração dos métodos de comunicação e conexão entre componentes, (ii) implementação de códigos para a coleta e transmissão dos dados de sensores para estações de

processamento, (iii) instalação de rádio modems que permitam a pilotagem da aeronave por piloto baseado em estação de controle terrestre, (iv) provisionamento de um *link* de vídeo para a pilotagem, e mais outros passos dependentes de cada operação e de cada plataforma. Desta maneira, o desenvolvimento de uma missão não possui, necessariamente, relação com outras missões, mesmo que elas compartilhem propriedades e recursos semelhantes, e isto tornam dispendiosos a implementação e o intercâmbio de aplicações neste tipo de veículo.

Por intermédio do proposto neste trabalho, as missões passam a ser gerenciadas pelo MOSA, que possui um processador de missões e que deixa o sistema de controle da aeronave totalmente independente da interpretação e do processamento da missão, como ilustrado pela Figura 1.1. O MOSA constitui um conjunto de sensores necessários para executar uma missão específica, gerenciados por um processador de missão. De acordo com as características da missão são determinados os sensores e é gerado um novo arranjo específico para aquela missão, ou seja, um novo MOSA (Pires et al., 2011). Com o passar do tempo, esta separação de responsabilidades entre MOSAs e VANTs pode permitir a construção de MOSAs com baixo custo (fator que dependerá dos sensores e dispositivos que os compõem e da escala de produção) e também contribuir para a multiplicação de operações (inclusive civis) com VANTs. O conceito MOSA tem, ainda, o potencial de transformar uma aeronave remotamente pilotada em um veículo aéreo autônomo.

Nem sempre a aeronave pode preencher todos os requisitos necessários para cumprir uma missão específica. Quando conectados, a aeronave e o MOSA deverão comunicar-se por meio do protocolo SSP trocando informações para decidir sobre o atendimento dos requisitos necessários para a realização da missão. Como resultado, o MOSA pode avaliar a missão como completamente viável, parcialmente viável ou inviável. Esta etapa é realizada sempre que um novo MOSA é conectado, ou sempre que uma nova missão for carregada e estabelecida no MOSA.

As missões podem ser adaptativas e algumas configurações podem ou deverão mudar durante a execução de uma missão, daí a importância de se ter uma estrutura dedicada como o MOSA. Ele é, agora, o responsável por decidir sobre a viabilidade da missão. O sistema da aeronave deve ser implementado somente para transportar esse arranjo de sensores, para realizar tarefas relacionadas ao controle do voo e que garantam a integridade da aeronave.

Os autores de (Thramboulidis et al., 2008) propõem em seu trabalho uma abordagem para a utilização da arquitetura orientada a serviços em sistemas embarcados convencionais. O artigo propõe um *framework* que facilita o desenvolvimento de sistemas embarcados, usando como serviços os recursos disponíveis. Os autores apresentam uma maneira fácil de integrar os componentes e os recursos desejados (por meio de técnica como a *plug and play*), que devem ser fornecidos pelas arquiteturas orientadas a serviços. Entretanto, essa abordagem não é apropriada para o cenário de sistemas críticos de tempo real,

como o [cenário] aqui utilizado. Sistemas embarcados críticos têm requisitos especiais, diferentemente de sistemas embarcados convencionais. O desempenho em tempo real é quase obrigatório e deve ser garantido sob qualquer circunstância. Alguns mecanismos de SOA como, por exemplo, a descoberta de serviços, tem potencial para comportamento não determinístico e não é compatível com os requisitos básicos de sistemas de tempo real.

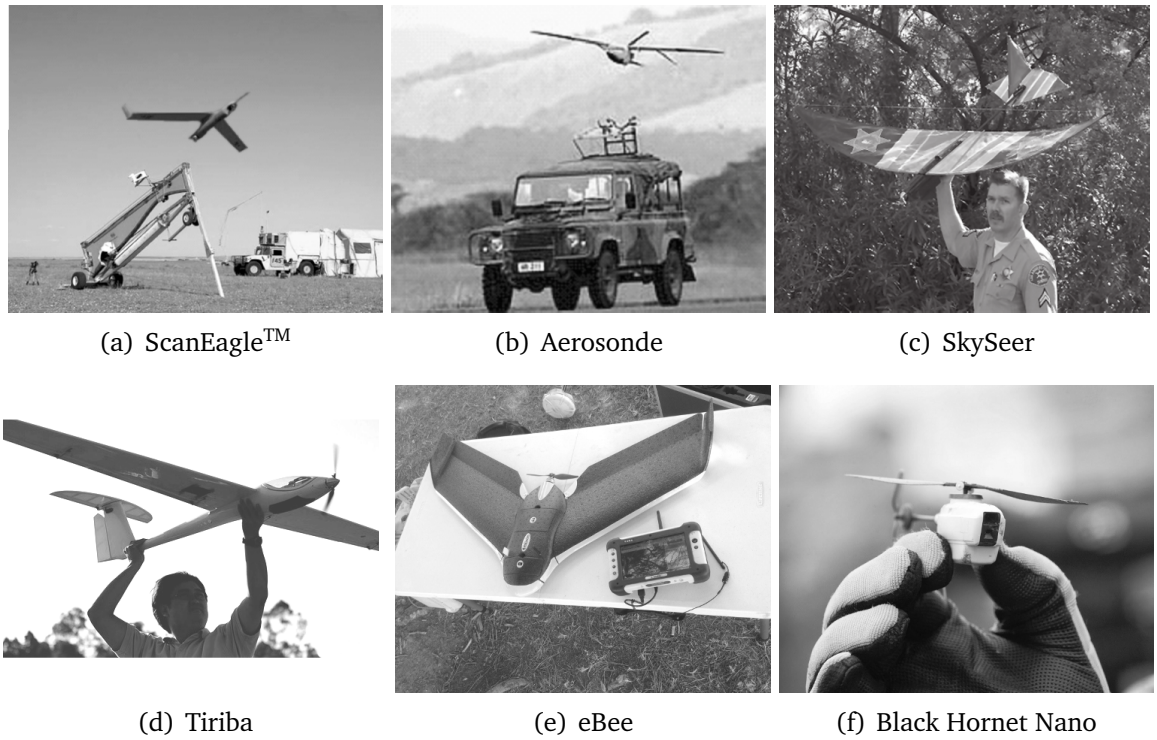
2.2 Aplicações e missões para VANTs

Os VANTs, assim como robôs em geral, variam muito em tamanho, forma e capacidades de voo. Alguns, como o Global Hawk (Figura 2.1), têm uma envergadura tão grande quanto a de um Boeing 737.

NASA Dryden Flight Research Center Photo Collection
http://www.dfrc.nasa.gov/Gallery/Photo/index.html
NASA Photo: ED07−0244−78 Date: December 3, 2007 Photo By: Tony Landis

Figura 2.1: Aeronave Global Hawk comprada pela NASA (agência espacial norte-americana), para coleta de dados de missões em ambientes como calotas polares e centro de vulcões (Fonte: (NASA, 2011)).

Outros, por não necessitarem de força ou dimensão suficientes para transportar uma *payload*, podem ser pequenos e leves o bastante para serem lançados com a mão, como é o caso do SkySeer e do brasileiro Tiriba, mostrados nas Figuras 2.2(c) e 2.2(d), respectivamente.



(a) ScanEagle™ (b) Aerosonde (c) SkySeer

(d) Tiriba (e) eBee (f) Black Hornet Nano

Figura 2.2: Exemplo de SANTS (adaptado de (Mica e Costello, 2008))

Encontra-se na literatura um número muito grande de modelos de VANTs com características variadas, quanto ao seu tamanho, tipo de asas, formato, mecanismo de propulsão, características de voo e as aplicações alvo de cada um. Por tamanha diversidade de modelos, os VANTs possuem potencial para uma infinidade de aplicações em pesquisas, em missões militares e para uso civil.

Ainda na literatura, foram encontradas várias referências sobre aplicações de veículos aéreos não tripulados para realizar diferentes tarefas. A maioria das aplicações encontradas (Jacques, 2002; Cooper e Goodrich, 2008; Pitre et al., 2009; George et al., 2011; van Willigen et al., 2011) se concentram em tarefas relacionadas à missões militares como busca, detecção, reconhecimento, rastreamento e ataque a alvos móveis e realocáveis, e utilizam sensores como câmeras fotográficas, de vídeo e de infravermelho, sensores de temperatura, ultravioleta, laser e detectores de reagentes químicos. Entretanto, essas tarefas não são tudo. Existem diversos outros contextos onde o uso desses sistemas pode trazer vantagens, como missões onde há um alto risco de colisão contra alvos ou abate de aeronave; locais com difícil acesso terrestre; missões com exposição de pessoas a situações perigosas; ambientes com alto risco de contaminação por radioatividade, supervisão de instalações de petróleo, contagem de árvores e animais em fazendas, detecção de patologias em plantações e muitas outras.

Os autores de (Merino et al., 2006) enumeraram em seu artigo algumas outras possíveis aplicações para UAVs, tais como:

- Aquisição de dados e imagens de alvos em áreas inacessíveis por meios terrestres;
- Localização de alvos;
- Rastreamento;
- Mapeamento de vias e construções;
- Cenários de desastres causados pelo homem ou pela natureza;
- Busca e resgate;
- Reforço na aplicação da lei;
- Vigilância e monitoramento do tráfego;
- Identificação de rodovias;
- Inspeção de linhas de transmissão de energia;
- Medições;
- Aplicações em agricultura de precisão;
- Cinematografia, e outras.

Cada aplicação diferente para os VANTs pode ser pensada como uma missão diferente. Isto se dá porque o contexto de uma missão é diferente do contexto de outra, visto que a lista de tarefas que serão executadas, a ordem de realização de cada uma, os requisitos de cada uma delas, bem como os parâmetros e os mapas dessas missões são diferentes em cada execução. Portanto, define-se como missão cada uma das possíveis aplicações para esses veículos, com suas tarefas, suas próprias características e seus próprios requisitos.

De acordo com o ponto de vista de missão aqui definido, observa-se a possibilidade da construção ou da formação de um *array* de sensores, ou seja, que diversos sensores possam ser agrupados, e então, um mesmo agrupamento de sensores possa ser parametrizado e ajustado com diferentes configurações para executar diferentes missões, e assim, cada instância desse agrupamento também poderá ser considerada distinta uma da outra.

Jacques, em (Jacques, 2002), cita exemplo de missões que utilizam VANTs para fazer busca, detecção, reconhecimento, ataque a alvos móveis e realocáveis com a utilização de câmeras coloridas e de infravermelho. Em (Pitre et al., 2009) VANTs são utilizados para detecção e rastreamento de alvos utilizando também uma câmera programada para tirar fotos a cada 30 segundos. Outras missões utilizando sensores como câmera de vídeo, sensor ultravioleta, laser, e sensor de temperatura são missões de busca e resgate em regiões selvagens (Cooper e Goodrich, 2008; Tso et al., 1999), busca e destruição de alvos de uma dada região (Drury et al., 2006), classificação, ataque, e verificação de danos infligidos (Karaman et al., 2009), rastreamento de objeto e seu reconhecimento em uma área (Maza et al., 2010), localização e detecção de fogo em florestas através da cooperação

de vários VANTs (Merino et al., 2006), controle e monitoramento de multidões, patrulha de fronteiras e monitoramento ambiental (Mullens et al., 2003).

Essas e outras diferentes aplicações que podem utilizar VANTs como veículo dependem de diferentes sensores que são acoplados à aeronave e que determinam a qualidade do resultado obtido com a execução da missão. Para cada uma delas um conjunto específico de sensores pode ser composto e parametrizado para prover o resultado esperado da missão, até mesmo em tempo real.

Por haver uma infinidade de variações de combinações de sensores para execução de missões, vê-se a necessidade e a vantagem de separar (ou isolar) do sistema de controle da aeronave a tarefa de processar a missão, de modo que, para tal, foi desenvolvido neste trabalho o protocolo SSP, detalhado no Capítulo 4, que permite a comunicação de um processador inteligente de missão – que por sua vez, estará interpretando e executando a missão – com o VANT incumbido do seu transporte.

2.3 Protocolos *plug and play*

Boa parte dos protocolos denominados *plug and play*, ou que possuem essa característica, está associada à automação industrial, principalmente aqueles destinados à automação doméstica (*smart homes*). Dentre esses protocolos destacam-se tecnologias como o Jini (SUN, 2011), a arquitetura UPnP (*Universal Plug-and-Play*) (Microsoft, 2000), o HomePnP (*Home Plug-and-Play*) (CEBus Industry Council, 1998; O'Driscoll, 2000) e o DLNA (*Digital Living Network Alliance*®) (DLNA Organization, 2006).

Jini é uma tecnologia desenvolvida pela Sun Microsystems™ que proporciona um mecanismo para que diversos dispositivos conectados em rede possam colaborar para a execução de tarefas e compartilhar recursos sem a necessidade de que o usuário final tenha que configurar a rede. A arquitetura Jini é distribuída e todos os dispositivos podem oferecer serviços e se comunicar. Em geral essa tecnologia é direcionada para dispositivos que executam Java, mas também permite que outros tipos de dispositivos possam ser conectados.

A arquitetura UPnP, uma arquitetura de *software* aberta e distribuída utilizada para a interligação de dispositivos por meio de uma rede, permite que as aplicações nos dispositivos conectados troquem informações e dados fazendo uso de uma arquitetura orientada a serviços. Tendo como base padrões como TCP/IP, UDP/IP e protocolos da internet, ela foi projetada para ser independente tanto de fabricante quanto do sistema operacional e da linguagem de programação de cada dispositivo, além de ser independente do meio físico. Com o UPnP, um dispositivo pode ingressar numa rede dinamicamente, obter um endereço IP, expor suas capacidades e descobrir automaticamente todos os outros dispositivos tam-

bém presentes, consequentemente, permitindo que os dispositivos comuniquem-se entre si diretamente.

O HomePnP buscava estabelecer um padrão no nível de aplicação por meio da definição do conteúdo das mensagens de controle que são trocadas entre dispositivos e controladores, o que torna possível descrever como diferentes produtos podem cooperar entre si. O objetivo principal deste protocolo era criar um padrão para interoperação de subsistemas de uma residência.

O projeto do padrão acabou sendo abandonado e, atualmente, o padrão correspondente é o DLNA.

O DLNA estabelece diretrizes baseadas nos padrões tecnológicos 802.3, 802.11, suíte de protocolo IPv4, HTTP, *UPnP Device Arquitecture* e *UPnP AV*, objetivando garantir a interoperabilidade entre eletrônicos conectados em uma rede doméstica, de modo que estes possam trocar arquivos de mídia entre si utilizando a rede em questão, ou seja, o usuário seria capaz de acessar, reproduzir e controlar a reprodução de seus arquivos de mídia armazenados em um computador ou em um armazenamento em rede através, por exemplo, de uma TV, de *tablets*, *smartphones*, entre outros, desde que esses se encontrem conectados na mesma sub-rede.

Apesar da diversidade de protocolos de arquitetura distribuída – e alguns até mesmo de arquitetura aberta – para dispositivos que oferecem e solicitam serviços poderem comunicar entre si, nenhum deles se aplica ao modelo de negócio proposto neste trabalho de mestrado. As soluções atuais têm em comum a proposta de distribuir a tarefa de controle entre os dispositivos e conectá-los por meio de uma arquitetura de rede adequada, permitindo a troca de informações diretamente entre eles para a execução das tarefas programadas, sem a necessidade de unidades de controle centralizadas, o que tem aumentado a eficácia e a confiabilidade de modo geral. Contudo, essas arquiteturas estão estruturadas para sustentar dados multimídias, um cenário distante dos SANTs.

A integração entre dispositivos e subsistemas de uma aeronave não tripulada é o ponto chave para a cooperação no cenário dos VANTs. Os requisitos de conexão e comunicação entre sensores, atuadores, placas de processamento de missão, *web servers* embarcados e sistemas de navegação e pilotagem automática são mais restritos que nas arquiteturas citadas nos parágrafos anteriores. Os desafios da comunicação e da integração para cooperação no cenário de veículos aéreos não tripulados requerem soluções mais adequadas e afinadas para esse ambiente.

Por isso propõe-se neste trabalho o desenvolvimento do protocolo SSP, cujo objetivo é viabilizar e padronizar a comunicação entre um MOSA e um VANT interconectados, para a cooperação mútua na execução de missões propostas para esse tipo de aeronave.

2.4 Considerações Finais

Este capítulo apresentou um levantamento feito sobre os trabalhos realizados com a mesma plataforma de veículo autônomo utilizado neste trabalho. Como não foi encontrado nenhum trabalho com o mesmo objetivo ou com objetivos próximos ao objetivo deste mestrado, apresentou-se outros trabalhos, relacionados com as técnicas com a metodologia utilizada aqui.

CAPÍTULO

3

Revisão bibliográfica

3.1 Sistema Embarcado

Um sistema embarcado é um sistema microprocessado onde seu processador é completamente encapsulado ou dedicado ao dispositivo ou sistema o qual ele monitora ou controla. Exemplos são telefones celulares, dispositivos de entretenimento digital doméstico, caixas eletrônicos bancários, impressoras, maquinário agrícola, computadores de bordo e centrais de controle de injeção e de frenagem em automóveis, placas de trânsito eletrônicas, equipamentos médicos, robôs industriais, veículos autônomos e, essencialmente, qualquer coisa com um microprocessador que não é considerada um "computador" propriamente dito mas que desempenha algum tipo de função usando computação.

Bilhões de microprocessadores embarcados foram vendidos nos últimos anos. Este mercado de sistemas embarcados tem crescido numa taxa extremamente alta não só em volume de produção, mas também em diversidade de aplicações. Hoje, os sistemas computacionais de propósito geral já foram superados, em números, pelos embarcados (Braga et al., 2011). Praticamente qualquer dispositivo doméstico, de entretenimento ou automotivo inclui um sistema embarcado, o que torna esses sistemas mais importantes economicamente que os sistemas de propósito geral.

Estes sistemas são considerados críticos quando eventos de falha podem acarretar perdas de vidas humanas ou perdas de ativos de alto valor. Em algumas aplicações, como na aviação, sistemas embarcados críticos devem apresentar taxas de falha baixas, como uma falha grave a cada 10^5 até 10^9 horas de operação (Trindade Jr et al., 2009).

Esses sistemas críticos tem requisitos especiais. Por questões como segurança e usabilidade, o desempenho em tempo real é quase obrigatório e deve ser garantido sob qualquer circunstância. Por outro lado, podem quase sempre ser segregados em seção de alta criticidade e seção de baixa criticidade (Rodrigues et al., 2011b), permitindo uma maior flexibilidade na implementação de códigos para esses sistemas.

3.2 Veículos Aéreos Não Tripulados

Uma aplicação típica de sistema embarcado crítico são os VANTs.

As capacidades de um VANT diferem das capacidades de uma aeronave tripulada em muitos aspectos. Um VANT pode atuar por períodos de tempo mais longos que um piloto conseguiria, com segurança, operar uma aeronave. Eles podem ser utilizados na agricultura de precisão, no monitoramento ambiental, na coleta de dados científicos e têm sido utilizados amplamente em missões militares (em questões relacionadas à segurança nacional).

Algumas aeronaves podem voar de forma autônoma, seguindo uma trajetória de voo pré-programada (baseada em um *grid* ou uma sequência de *waypoints*) (Trindade Jr et al., 2002, 2010), enquanto outras voam recebendo comandos a partir de estações de controle terrestres operadas por pilotos. A estação de controle pode ser implementada em *smartphones*, *tablets*, *notebooks* ou redes de estações de trabalho (estações de controle distribuídas).

O avião pode variar não apenas em tamanho, mas também na forma, no tipo de propulsão e no desempenho. A interface homem-máquina (ou homem-robô) 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 cumprir a missão destinada ao sistema.

A falta de padronização de uma metodologia para categorizar esse tipo de aeronave cria uma dificuldade para se correlacionar VANTs diferentes. Essa ausência de normas internacionalmente aceitas para a classificação permite que cada país desenvolva suas próprias regras para essa categorização.

Em muitas das vezes os VANTs encontram-se classificados segundo seus recursos, tamanhos e capacidades. A fim de apresentar uma classificação geral destas aeronaves citam-se aqui seis categorias funcionais e quatro categorias relacionadas à capacidade e à finalidade das aeronaves. As categorias funcionais são:

- Alvo: aeronaves que servem como alvo para outra aeronave ou um míssil;
- Reconhecimento: aviões destinados a prover inteligência ao campo de batalha;

- Combate: aviões com capacidade de ataque para missões de alto risco (combates aéreos de veículos não tripulados);
- Logística: VANTs projetados especificamente para o transporte de cargas e operações logísticas;
- Pesquisa e Desenvolvimento: VANTs utilizados para desenvolver tecnologias que serão utilizadas em outras categorias de VANTS;
- Civil e Comercial: VANTs especificamente projetados para aplicações civis e comerciais.

As quatro classes não funcionais são (Clapper Jr et al., 2007):

- Pequenas: peso bruto de decolagem inferior a 55Kg;
- Táticas: peso bruto de decolagem entre 55 e 1320Kg;
- Theater: peso bruto de decolagem superior a 1320Kg;
- Combate: projetadas desde o início como uma plataforma de ataque com bombas, armas e equipamentos para sobrevivência. Peso bruto de decolagem maior que 1320Kg.

Segundo a U.S. Army (U.S. Army, 2010), os VANTs podem ser classificados em termos de alcance, altitude de voo e velocidade, conforme a Tabela 3.1. Nessa tabela foram incluídos os VANTs brasileiros Tiriba, ARARA (Trindade Jr et al., 2012) e Nauru.

(a) Tiriba (b) ARARA (c) Nauru

Figura 3.1: VANTs brasileiros

3.3 Sistema de Aeronave Não Tripulada

A aeronave sozinha não consegue representar todo o sistema que está envolvido na execução de missões inteligentes que utilizam veículos não tripulados. Por isso, o termo SANT (Sistema de Aeronave Não Tripulada), do inglês UAS (*Unmanned Aircraft System*),

Tabela 3.1: Classificação de VANTs segundo *U.S. Army* (traduzido e adaptado de (U.S. Army, 2010))

Categoria	Peso máximo de decolagem [lbs]	Altitude de voo [ft]	Velocidade [KIAS]	Representantes da categoria
Grupo 1	0-20	< 1200 AGL	100kts	WASP III, Future Combat, System Class I, TACMAV RQ- 14A/B, BUSTER, BATCAM, RQ-11B/C, FPASS, RQ-16A, Pointer, Aqua/Terra Puma, Tiriba®.
Grupo 2	21-55	< 3500 AGL	< 250kts	Vehicle Craft, Unmanned Aircraft System, ScanEagle, Silver Fox, Aerosonde, ARARA.
Grupo 3	< 1320	< 18000 MSL	< 250kts	RQ-7B, RQ-15, STUAS, XPV-1, XPV-2, SARVant.
Grupo 4	> 1320	< 18000 MSL	Qualquer velocidade	MQ-5B, MQ-8B, MQ-1A/B/C, A-160.
Grupo 5	> 1320	> 18000 MSL	Qualquer velocidade	MQ-9A, RQ-4, RQ-4N, Global Observer, N-UCAS.

AGL – *Above Ground Level* (Acima do nível do solo)
MSL – *Mean Sea Level* (Nível médio do mar)
KIAS – *Knot Indicated Airspeed* - Velocidade indicada do ar

foi adotado pela americana FAA (*Federal Aviation Administration*) e pela comunidade acadêmica internacional para designar sistemas que incluem não apenas os aviões não tripulados, mas todos os elementos associados, tais como a *payload*, a estação de controle terrestre, a infraestrutura de suporte e os *links* de comunicação (Mica e Costello, 2008), como ilustrado pela Figura 3.2.

Um SANT é composto pelos seguintes elementos principais (Mica e Costello, 2008; Trindade Jr et al., 2012):

- **Aeronave não tripulada**, composta principalmente pela estrutura, grupo motopropulsor, acionadores, sistema de guiamento e controle e subsistemas de comunicação;

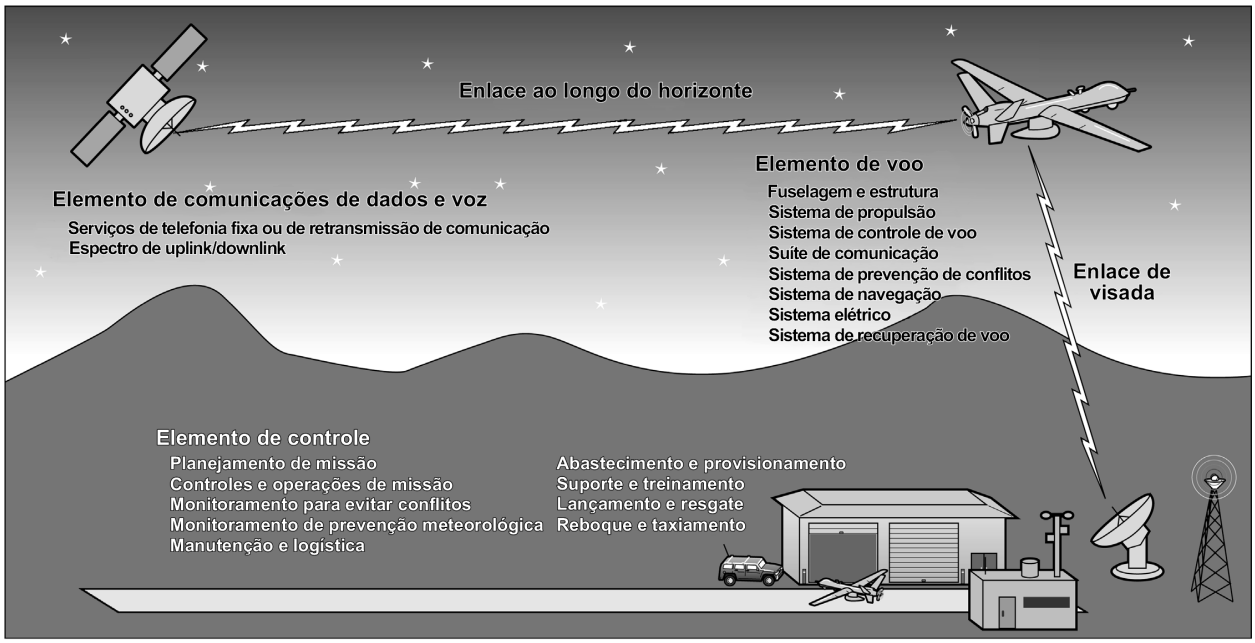


Figura 3.2: Conceito de um SANT (traduzido de (Mica e Costello, 2008))

- **Centro de monitoramento e controle em solo**, composto principalmente por estações de trabalho com a finalidade de Controle e Monitoramento, Comunicação e Controle de *payload*, normalmente constituída por sensores de imageamento no espectro visível ou outros;
- **Antenas de comunicação**, normalmente com opção de rastreamento da aeronave para aumento de alcance e largura da banda de comunicação;
- **Dispositivos auxiliares para pouso e lançamento**, incluindo catapultas, foguetes, trem de pouso ejetável e redes de recuperação, entre outros;
- ***Payload***, normalmente sensores, embora possam englobar armamento e outros dispositivos.

Afim de propor uma compreensão mais ampla das partes componentes de um Sistema de VANT, um modelo de referência foi proposto no trabalho (Rodrigues et al., 2011b), ilustrado pela Figura 3.3.

Um modelo de referência procura oferecer uma semântica comum que pode ser usada de forma não ambígua através e entre implementações diferentes. Nesse sentido, um modelo de referência tem a intenção de oferecer um alto nível de coisas comuns.

Na literatura, os termos modelo arquitetural, arquiteturas, arquiteturas de referência e modelos de referência são tratados como sinônimos, apesar de terem significados distintos, mesmo que em alguns casos essa diferença seja sutil.

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. Por outro

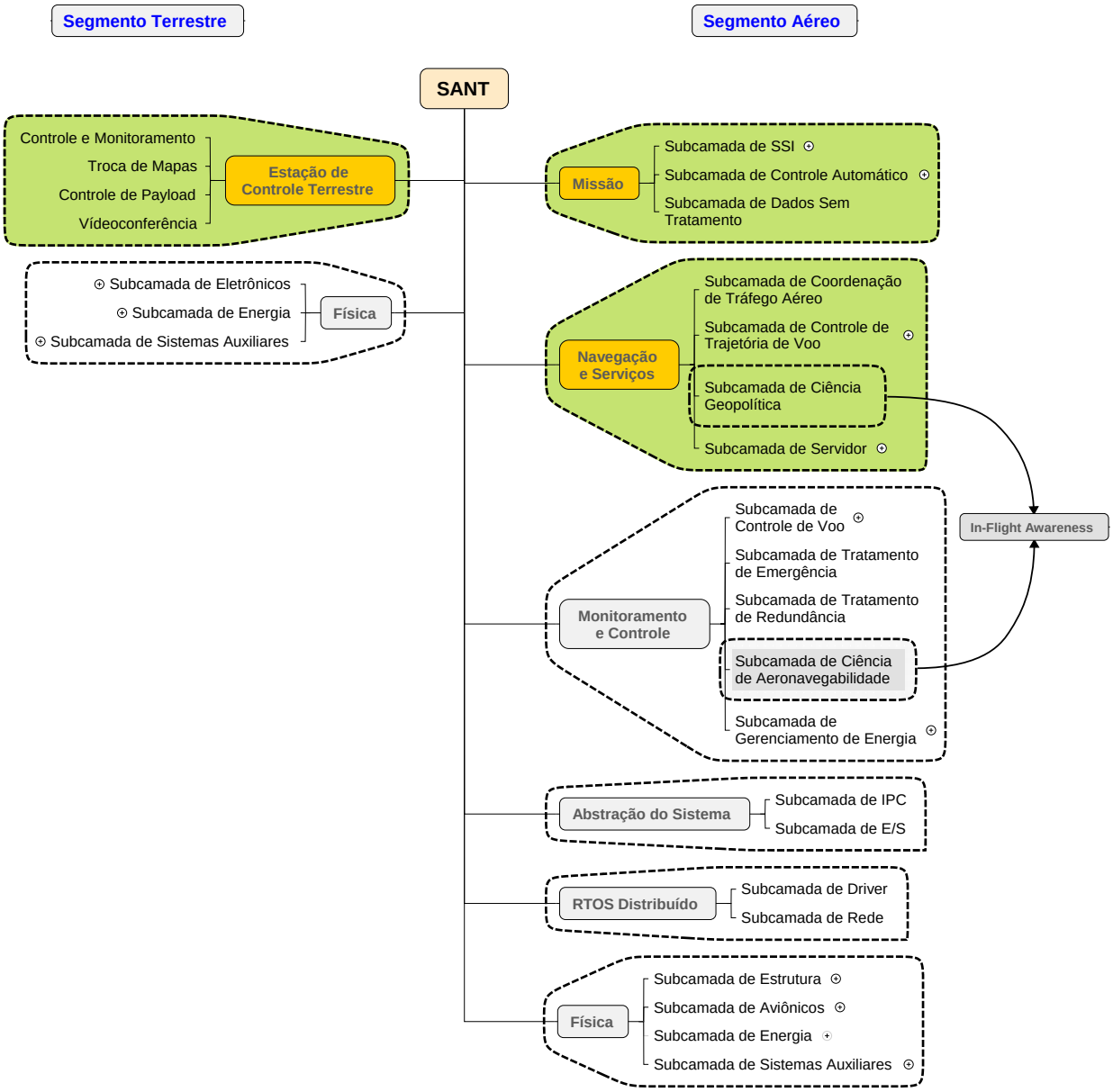


Figura 3.3: Modelo de referência para SANTs (traduzido de (Rodrigues et al., 2011b))

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 de um dado subsistema 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. O termo modelo de arquitetura utilizado neste trabalho reflete exatamente essa última afirmação: incorporar o objetivo básico e as ideias do sistema.

No modelo ilustrado pela Figura 3.3, os componentes de um SANT podem ser divididos em um segmento aéreo e um segmento terrestre. O segmento aéreo é hierarquicamente composto pela (i) camada física, (ii) camada distribuída de RTOS (*Real Time Operating*

System), (iii) camada de abstração do sistema, (iv) camada de monitoramento e controle, (v) camada de navegação e serviços e (vi) camada de missão. O segmento terrestre é dividido em (i) uma camada física e (ii) uma camada de estação de controle terrestre.

O NIST (*U.S. National Institute of Standards and Technologies*) também provê um modelo de referência para VANTs (Albus et al., 2002). Neste padrão específico, o modelo de referência foi proposto para especificar as regras militares, usos e comandos de forma compreensível e intuitiva para um comandante humano.

A Figura 3.3 apresenta uma visão detalhada e em camadas, ilustrando as entidades e suas relações, sendo uma abordagem diferente da apresentada pelo NIST em (Albus et al., 2002).

Uma modelagem em camadas provê flexibilidade ao sistema, permitindo que ele seja dividido em subsistemas que podem ter implementações independentes, e no caso do cenário de sistemas embarcados complexos, permite também que as partes que compõem esses sistemas sejam divididas em níveis de criticidade diferentes. Por exemplo, as partes destacadas em verde na figura supracitada são camadas do sistema com baixa criticidade, onde pode-se fazer o uso da arquitetura orientada a serviços para comunicação entre processos que conversam no mesmo nível (Rodrigues et al., 2011a,b; Pires et al., 2011).

3.4 Projeto de Protocolos

Um protocolo define uma interação entre subsistemas não adjacentes. É uma convenção que controla e possibilita uma conexão, comunicação ou transferência de dados entre dois sistemas computacionais. Protocolos de rede operam entre partes distantes do sistema, fazendo uso da habilidade da rede em carregar mensagens.

Protocolos também podem ser usados para descrever os procedimentos para a troca de informação entre processos que estejam não apenas num ambiente de rede, mas também entre sistemas com multiprocessadores para controlar a interação de processos paralelos, em aplicações de tempo real para controlar um número de diferentes dispositivos, e em outros sistemas onde não há relação de tempo fixo entre uma ocorrência de um evento e a ação implicada por ele (Tarnay, 1991). Eles podem ser implementados pelo hardware, pelo software ou por uma combinação dos dois.

Um protocolo é definido em termos de suas sintaxe e semântica e deve ser implementado entre dois processos, descrevendo a ação de cada processo em forma de estados ou eventos. Quando uma definição do protocolo tiver sido estabelecida com precisão, ela deve ser testada quanto à sua correção e coerência (Tarnay, 1991).

O projeto de um protocolo inicia-se pela etapa de especificação formal, com a utilização de ferramentas formais para descrição das atividades do protocolo, garantindo maior

precisão e menor possibilidades de ambiguidades. A fase da especificação permite ao desenvolvedor preparar um modelo abstrato do protocolo para testes e análises. Tarnay define esta etapa de descrição formal em (Tarnay, 1991) como sendo a base do projeto, da implementação, da verificação e do teste de um protocolo.

Inúmeras organizações de padronização tais como *International Organization for Standardization* (ISO), *International Telecommunications Union* (ITU), NIST etc, desenvolveram técnicas concretas para a realização de descrições formais (*Formal Description Techniques* – FDTs) (ISO, 1989, 1997; ITU, 1999). Um princípio evidente é que as FDTs tem validade geral (Tarnay, 1991) e, de acordo com (Turner, 1993), elas foram criadas com o objetivo de assegurar especificações claras, concisas e não ambíguas, além de possuírem características como completude, consistência e tratabilidade, a fim de garantir que implementações estejam em conformidade com especificações.

As FDTs abrangem o uso de linguagens de programação ou de modelos de transição para fazer a descrição formal de sistemas.

3.4.1 Especificação de sistemas

Linguagens de programação

A utilização de linguagens de programação para definir as características de um sistema aproveita-se da proximidade ou da estreita relação com a especificação deste sistema utilizando um algoritmo. De fato, um protocolo é um algoritmo e pode ser precisamente definido por meio de uma linguagem de programação de alto nível (Moura et al., 1986).

Essa técnica é utilizada em modelos de protocolos muito complexos, com explosão de estados[1] inevitáveis ou quando nenhuma outra técnica de solução para tal explosão possa ser utilizada. Ela possui a vantagem da descrição explícita das variáveis e parâmetros, o que não pode ser feito em descrições com modelos de transição.

Modelos de transição

Linguagens formais

Uma linguagem formal é um conjunto de sentenças, e uma sentença é uma sequência finita (ou *string*) de elementos, cada qual vindo de um vocabulário finito. Por exemplo, dado que um vocabulário V da linguagem L é o conjunto de dois membros $\{a, b\}$, L deve ter entre seus membros não só a, b, ab e ba, mas também aa, aba, bba, etc. O tamanho da

[1] Explosão de estados é um dos problemas inerentes da verificação de modelos complexos, onde a árvore de alcançabilidade do modelo torna-se extremamente grande, tanto em largura quanto em profundidade, a ponto de extinguir a memória disponível para se realizar a análise.

string é determinado pelo número de elementos que a compõem. *Strings* podem ter tamanhos finitos ou infinitos, mas as sentenças possuem, necessariamente, tamanhos finitos.

Uma linguagem pode ser definida por uma gramática particular como um conjunto de todas as sentenças que ela gera. Uma gramática, por sua vez, pode ser definida pelo conjunto de símbolos terminais e não terminais e suas regras de aplicação.

Diagramas de estados

Ações de processos também podem ser descritas em termos de estados de processo e eventos, que são a chegada de mensagens nestes processos e que causam a transição entre estados (Figura 3.4).

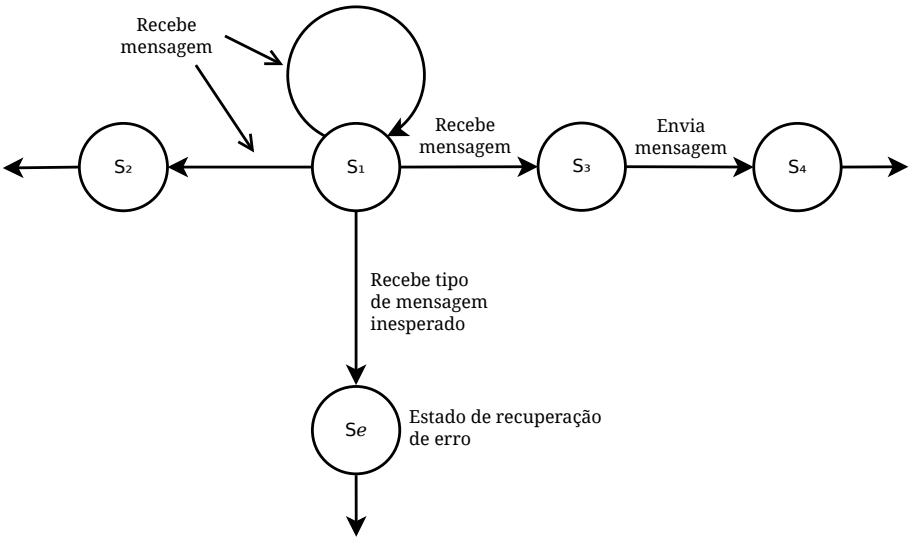


Figura 3.4: Exemplo de estados de um protocolo. Cada entrada em um estado funciona como um estímulo, gerando uma transição para um estado seguinte.

Quando um processo está em um estado particular ele define um subconjunto de mensagens que podem ser recebidas e a ação que deve ser tomada na recepção de qualquer uma delas. Ações devem ser definidas para todas as mensagens recebidas. Se uma outra mensagem desconhecida – neste estado – é recebida então a sincronização foi perdida, e é necessário entrar na parte de recuperação de erro do protocolo. Após receber uma mensagem de uma das portas de entrada, o processo pode entrar em um estado intermediário durante o qual ele, além de processar tarefas internas, formata e despacha uma mensagem de saída.

A descrição da implementação de um protocolo por meio de diagrama de estado torna-se difícil no caso de protocolos complicados, que possuam um número muito grande de estados. Nestes casos, é mais apropriado o uso de linguagens de alto nível.

Modelos mistos

Os modelos mistos ou híbridos utilizam, para a mesma especificação, os dois modelos citados anteriormente, o que permite se beneficiar das vantagens de cada um deles. Do modelo de transição utiliza-se os aspectos de controle e a facilidade oferecida pela visualização das mudanças de estados. Das linguagens formais aproveita-se da possibilidade de especificação explícita de variáveis e parâmetros.

No projeto do protocolo SSP o método adotado foi o de modelagem mista, como será visto no Capítulo 5. Isto acarretou em um aumento no trabalho a ser executado, contudo, perante o resultado que se buscava, foi necessário utilizar uma ferramenta de validação formal e isto requereu a modelagem do protocolo tanto utilizando a linguagem de alto nível da ferramenta quanto o diagrama de estados do protocolo.

A fase seguinte no projeto de protocolos é a verificação, onde avalia-se a presença de características que podem causar futuros erros, tais como o *deadlock*.

3.4.2 Verificação da especificação

A etapa de verificação de um modelo de protocolo visa garantir a correção e a completude do modelo especificado na etapa anterior.

Existem basicamente duas abordagens para a verificação formal.

A primeira é a verificação de modelos, que consiste numa exploração sistematicamente exaustiva do modelo matemático, o que é possível para modelos finitos, mas também para modelos infinitos nos quais conjuntos finitos de estados podem ser representados. Isso geralmente consiste na exploração de todos os estados e transições do modelo, por meio do uso de técnicas de abstração para considerar grupos de estados numa única operação, reduzindo, assim, o tempo de processamento.

A segunda abordagem é a inferência lógica, que consiste na utilização de uma formalização matemática do sistema, por meio do uso de *software* de prova de teoremas. Esse processo é geralmente automatizado parcialmente, dependendo do entendimento manual do sistema para a validação.

As propriedades a serem verificadas são (Ferreira, 2005; Carvalho, 2009): (i) Ausência de *deadlocks*, (ii) Completude, (iii) Atividade ou Realização de Progresso, (iv) Terminação, (v) Correção, (vi) Minimidade e (vii) Estabilidade.

Para que tais propriedades de um modelo de transição possam ser verificadas a técnica mais adotada é a análise de alcançabilidade, onde uma árvore de alcançabilidade é gerada manual ou automaticamente com todos os estados possíveis de serem alcançados a partir de um estado inicial (Moecke e Farines, 1992; Carvalho, 2009).

A verificação adotada no projeto do protocolo SSP utilizou a inferência lógica para validar o protocolo implementado. A ferramenta utilizada para a verificação do modelo

utiliza regras da Lógica Temporal (explicada na Seção 5.1) inseridas manualmente para fazer a análise de alcançabilidade dos estados do modelo, conforme requer cada regra inserida.

3.4.3 Testes da implementação

O teste da implementação de um protocolo é feito para determinar se tal implementação está de acordo com a especificação feita no início do projeto. Caso o protocolo implementado execute as funções e forneça os serviços assim como foram especificados, então, essa implementação é aprovada no teste.

O teste de *softwares* é planejado em termos de cenários, onde em cada cenário é definida uma seleção de entradas que são fornecidas à implementação para serem processadas. As saídas geradas são, então, comparadas com os valores esperados e, com base nessa comparação, define-se o resultado do teste. Em implementações de protocolos de comunicação, comumente testa-se os três pontos que seguem: (i) as respostas corretas a pedidos válidos de serviços, (ii) comportamento com implementações do mesmo protocolo em sistemas diferentes, (iii) rejeição de erros de pedidos inválidos de serviços (Cordeiro et al., 2003).

É interessante que os testes sejam realizados no ambiente onde a implementação será utilizada, sob condições reais de operação. Se, no caso de protocolos de comunicação, a rede não estiver disponível pode-se utilizar, para a realização dos testes, um simulador que reflita as condições de operação da rede.

Para executar testes de implementação do protocolo SSP, foi montado um ambiente de experimentação para simular o ambiente de operação do protocolo. Foram codificadas duas aplicações em linguagem Java que se comunicavam com primitivas distintas. Cada aplicação foi executada em um computador diferente, que representavam o processador de missão e a aeronave comunicantes. Esses computadores foram interligados por placas de prototipação Arduino, onde o modelo do SSP foi codificado para operar e ser avaliado.

3.5 Considerações Finais

Neste capítulo foi mostrado uma revisão bibliográfica sobre todos os assuntos abordados neste trabalho. Explicou-se a diferença entre VANT e SANT, e como se dá o processo de design de um protocolo, desde a etapa de especificação até a etapa de teste da implementação. O emprego destas etapas do projeto é realizado no próximo capítulo, onde é apresentado todo o projeto do protocolo SSP.

CAPÍTULO

4

O Projeto do Protocolo SSP

O projeto do protocolo SSP foi focado na simplicidade e, uma vez que ele é apenas um mecanismo de interfaceamento, ele provê uma camada simples e que não deve causar um *overhead*[1] na comunicação entre as entidades. Ele propicia uma conexão, verifica a viabilidade da execução de uma missão, viabiliza a comunicação entre as entidades durante a execução da missão e encerra essa sessão de cooperação, não intervindo nos dados trafegados, o que o torna um protocolo facilmente adaptável às outras arquiteturas de veículos autônomos.

4.1 Arquitetura

A arquitetura do protocolo SSP é formada por quatro elementos: um processador de missão, um *middleware* SSP para esse processador, uma aeronave não tripulada e um *middleware* SSP para essa aeronave, como ilustrado pela Figura 4.1.

Processador de missão – MOSA A função do processador de missão é, literalmente, processar as tarefas de uma missão, lendo e interpretando os dados vindos de sensores, acoplados a si. Este processador trabalha enviando requisições de deslocamento, de serviços ou de propriedades do voo à aeronave;

[1] *overhead* é geralmente considerado qualquer processamento ou armazenamento em excesso, seja de tempo de computação, de memória, de largura de banda ou qualquer outro recurso que seja requerido para ser utilizado ou gasto para executar uma determinada tarefa. Como consequência, pode piorar o desempenho do sistema que sofreu o *overhead*.

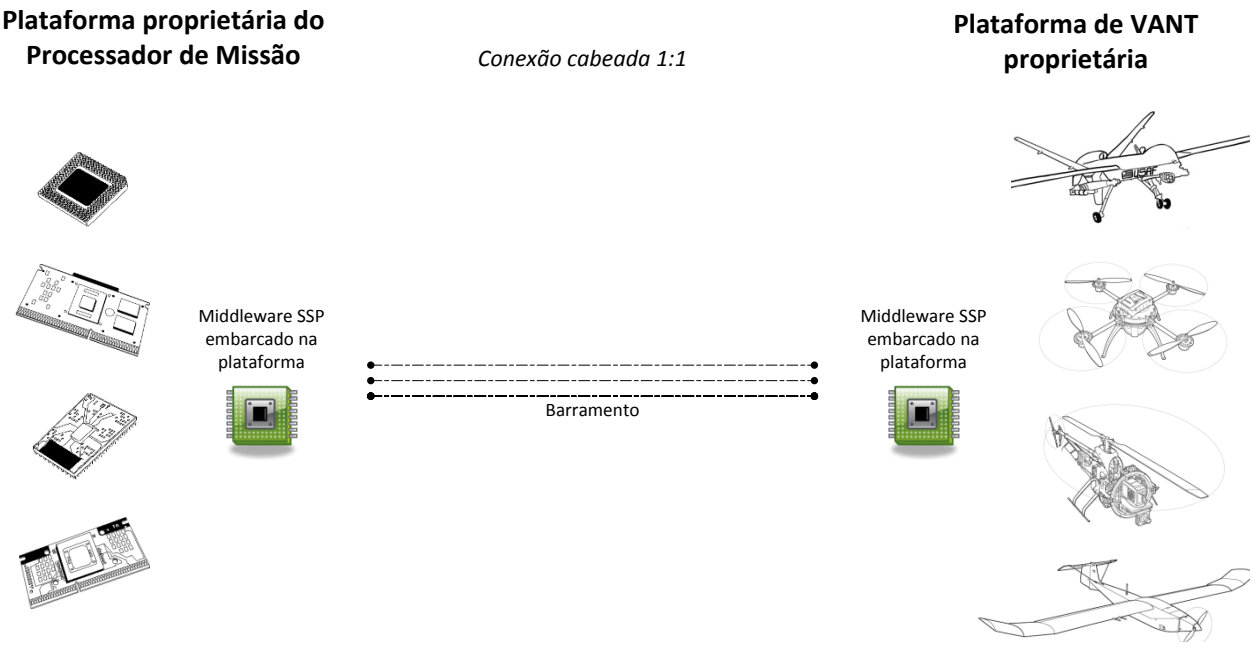


Figura 4.1: Arquitetura do ambiente do SSP

***Middleware* MOSA** O papel do *middleware* SSP é permitir que diferentes implementações de MOSAs interajam com diferentes implementações de VANTs. O middleware do MOSA é responsável por traduzir as mensagens proprietárias do MOSA para o formato das mensagens do SSP, e vice-versa;

***Middleware* VANT** O papel do *middleware* SSP integrado ao VANT é traduzir as mensagens SSP entrantes para o formato proprietário compreendido pelo avião, bem como fazer o caminho inverso, que é a tradução das mensagens do avião para o formato das mensagens do SSP;

VANT A função do avião não tripulado é receber e responder às requisições do MOSA. Ele deve atender a funcionalidade de deslocamento no ar e pode oferecer informações sobre suas características e sobre outros possíveis serviços, como informações sobre o voo, por exemplo.

4.2 Estrutura das Mensagens

O serviço a ser fornecido por um protocolo de comunicação é especificado por um conjunto de operações – ou primitivas – de serviço disponível para que uma entidade possa acessá-lo. Uma primitiva indica a execução de alguma ação ou a geração de uma notificação sobre uma ação executada por uma entidade parceira.

As primitivas são uma representação abstrata da interação entre duas entidades sobre um serviço, e são ordinariamente divididas em quatro classes (**?**Hura e Singhal, 2001; Bonaventure, 2012), como apresentado na Tabela 4.1:

Tabela 4.1: Primitivas básicas de um protocolo de comunicação

Primitiva	Significado
Request	Uma entidade solicita uma ação à outra
Indication	Uma entidade é informada sobre um evento
Response	Uma entidade responde a um evento
Confirm	Resposta a uma solicitação anterior da entidade

As primitivas do protocolo SSP são classificadas em: Negociação, Execução, Pouso, Encerramento e Cancelamento. Na Tabela 4.2 são apresentadas conforme sua classificação e seguindo as etapas de comunicação em que são utilizadas.

Tabela 4.2: Primitivas do protocolo SSP

Etapa	Primitiva	Significado
1. Negociação	REQ	Esta primitiva é utilizada pelo *middleware* do processador de missão para requisitar características de operação da aeronave. Ela só é aceita na primeira etapa da comunicação.
	RET	Esta primitiva é utilizada pelo *middleware* do avião para responder uma solicitação `REQ`. Esta primitiva também só é aceita na primeira etapa da comunicação.
	SEND	A primitiva `SEND` é utilizada para transportar mensagens entre processos internos das entidades, como previsto no cenário de operação. Na etapa de *Negociação*, esta primitiva só deve ser usada momentaneamente, como no caso das operações da arquitetura SOA de divulgação, descoberta e inscrição em serviços, por exemplo. Dados a serem trocados entre processos internos ao processador de missão e à aeronave devem trafegar como *payload* de uma primitiva `SEND`, em ambos os sentidos.

Tabela 4.2: Primitivas do protocolo SSP (continuação)

Etapa	Primitiva	Significado
1. Negociação	ACK	A primitiva `ACK` é utilizada nesta etapa para um *middleware* confirmar o correto recebimento de um `SEND`. Esta confirmação deve acontecer após cada mensagem `SEND`, indicando ao *middleware* remetente que a mensagem foi recebida e repassada ao processo destinatário.
2. Decolagem	TAKEOFF	`TAKEOFF` é a primitiva utilizada pelo *middleware* do processador de missão para notificar ao *middleware* do avião que a negociação já terminou e que a aeronave precisa decolar para que um deslocamento seja requisitado. Ao decolar a aeronave deve manter-se em torno do ou sob o *waypoint* `<Home>`, informado como *payload* da mensagem.
	READY	A primitiva `READY` é utilizada pelo *middleware* do avião para notificar ao *middleware* do processador de missão que a decolagem foi realizada, que a aeronave encontra-se no ar e presente no *waypoint* `<Home>`, como requisitado na mensagem anterior.
3. Execução	GOTO	A primitiva `GOTO` marca a mudança de etapas da cooperação. Uma vez que a primeira mensagem `GOTO` é recebida, dá-se a missão como iniciada. Esta primitiva é utilizada durante toda a execução da missão e é enviada no sentido processador $\rightarrow$ aeronave, solicitando o deslocamento para um *waypoint* dado como *payload* da mensagem.
	SEND	Na etapa de *Execução*, esta primitiva pode ser usada a todo momento, para troca de mensagens entre processos cliente e provedor de *webservices*. Novamente, o conteúdo das mensagens deve ser encapsulado e enviado no campo *payload* da mensagem.
	NOTIFY	Primitiva enviada pelo *middleware* da aeronave para indicar ao *middleware* do processador que um *waypoint* solicitado foi alcançado.
	ACK	Esta primitiva é utilizada, pelo *middleware* destinatário, em toda a etapa de execução após cada mensagem `GOTO`, `SEND` e `NOTIFY` recebida, para notificar ao remetente o correto recebimento destas mensagens.

Tabela 4.2: Primitivas do protocolo SSP (continuação)

Etapa	Primitiva	Significado
4. Pouso	LAND	A primitiva `LAND` é utilizada para requisitar o pouso da aeronave, e como *payload* da mensagem ela fornece um *waypoint* e um *heading* da pista para pouso.
	ACK	Uma mensagem `ACK` deve ser respondida pela aeronave ao confirmar a instrução de pouso feita pelo processador de missão.
5. Encerramento	CLOSE	A primitiva `CLOSE` é utilizada para terminar a comunicação entre as entidades. Ela é enviada no sentido processador → aeronave.
5. Encerramento	ACK	A primitiva `ACK` é utilizada nesta etapa pela aeronave para confirmar a identificação de uma mensagem `CLOSE` e encerrar a comunicação.
6. Cancelamento	ABORT	A primitiva `ABORT` só é utilizada pela aeronave em algum caso extremo onde o sistema de controle da própria aeronave – que possui maior preempção – precisa assumir o comando dela. Ao emitir uma mensagem `ABORT` a aeronave cancela qualquer operação em andamento e encerra imediatamente a comunicação com o processador de missão, não sendo necessário o envio de uma mensagem de confirmação por parte do processador. Este tipo de mensagem pode ser emitida em momentos tais como baixo nível de combustível ou carga de bateria do motor, ou conforme tenha sido implementado pelo proprietário da arquitetura.

Com essas primitivas é possível então que o MOSA se comunique com a aeronave ou com qualquer outra plataforma que possua o *middleware* SSP e vice-versa.

4.3 Comunicação

O protocolo SSP é um protocolo da camada de aplicação e o seu desenvolvimento se baseou em conceitos simples de protocolos já consolidados e de mesma camada. As mensagens SSP possuem o formato `cabeçalho | payload`, onde o primeiro campo representa o cabeçalho do protocolo, que inclui a primitiva de serviço daquela mensagem, e o segundo campo representa os dados do usuário que a mensagem carrega.

Uma mensagem SSP válida sempre deve conter uma das primitivas conhecidas e o recebimento de todas as mensagens deve ser notificado. Avaliado o cabeçalho da mensagem e, caso a primitiva não seja reconhecida, o protocolo foi modelado para simplesmente ignorar o pacote inteiro.

Uma perspectiva global da comunicação sobre SSP entre as duas entidades desse cenário foi descrita por meio de um diagrama de estados e um diagrama de troca de mensagens, representados, respectivamente, nas Figuras 4.2 e 4.3.

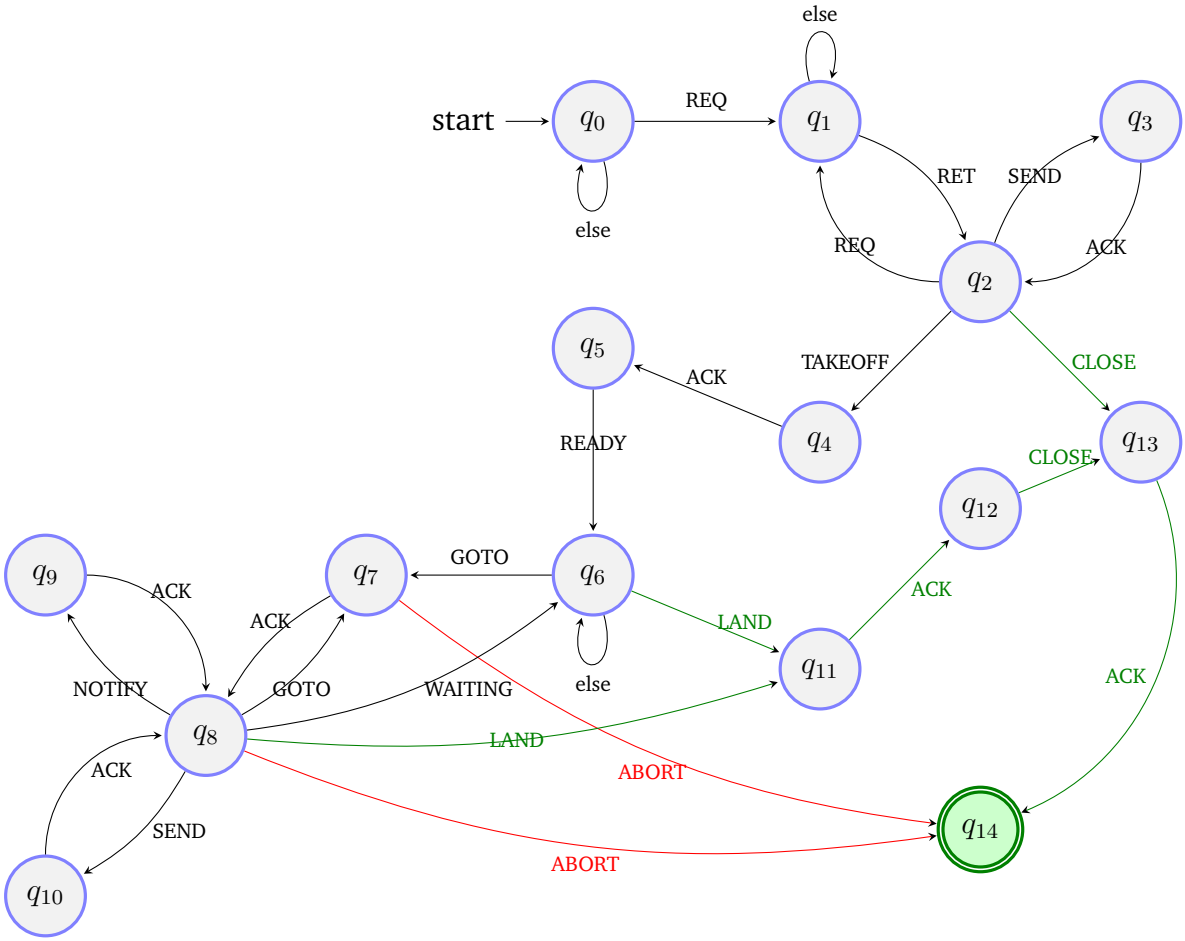


Figura 4.2: Diagrama de estados do protocolo SSP, sob uma perspectiva global da comunicação. O estado $q0$ é o estado inicial e o $q14$ é o estado final. Observa-se a presença de transições `else` para ignorar pacotes não reconhecidos.

No diagrama de estados, cada aresta faz a transição de um estado para outro por meio da emissão de uma mensagem. A emissão de mensagens ocorre de modo alternado entre as entidades comunicantes. Deste modo, a mesma entidade nunca provoca duas transições sucessivas de estado.

Como pode ser visto no diagrama de mensagens (Figura 4.3), a cooperação entre as entidades inicia-se, após a conexão, com a mensagem emitida pelo processador de missão à aeronave. A cooperação se dá em três etapas sequenciais: (i) a negociação dos parâmetros

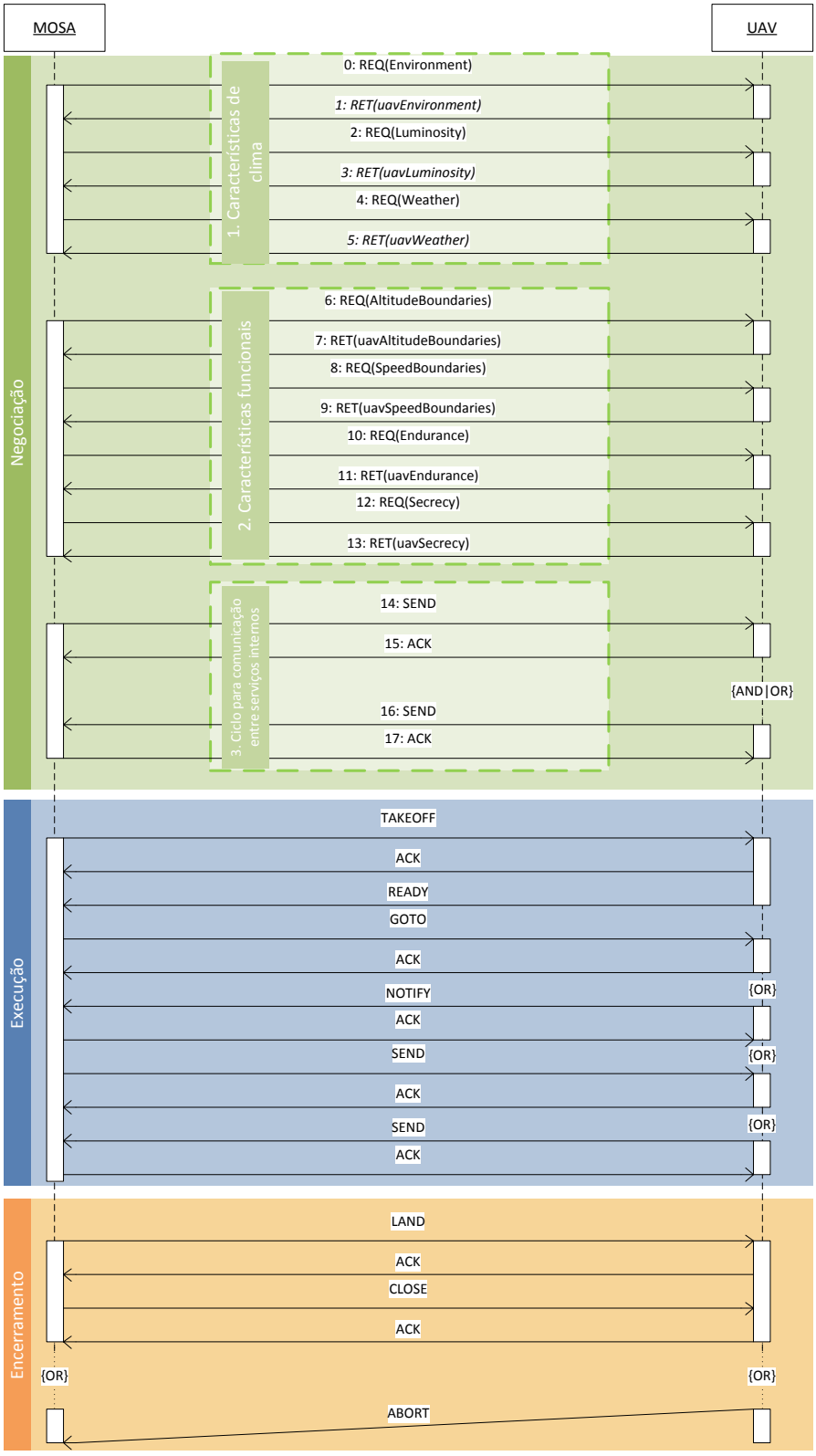


Figura 4.3: Diagrama de Sequência de Mensagens SSP trocadas entre um processador de missão e uma aeronave não tripulada. Observa-se a presença dos condicionais {AND} e {OR} nas etapas de Negociação de Serviços, durante toda a Execução da Missão e no Encerramento da Missão.

da missão e a definição da sua viabilidade, (ii) o processamento da respectiva missão e (iii) o encerramento da missão e término da comunicação.

A primeira etapa da conversação entre as entidades é nomeada Negociação, onde são feitas requisições e disponibilizações das características da aeronave e dos *webservices* internos disponíveis. As transições da Negociação vão do estado *q0* ao estado *q3*, na Figura 4.2.

Após terminada a fase de Negociação, caso o processador tenha definido a missão como viável de ser executada com a aeronave associada, então, prossegue-se para a segunda etapa, a fase de Execução. Para isso, o processador envia uma mensagem `TAKEOFF` informando um *waypoint* denominado *Home*, de onde a aeronave deve partir para a execução da missão. Nesta etapa a aeronave fornece a capacidade de deslocamento e, se implementado, alguns serviços ao processador de missão. As transições da fase de Execução vão do estado *q2* ao estado *q10*.

Terminada a execução da missão pelo processador de missão, a aeronave ainda encontra-se no ar e precisa pousar. Essa operação deve ser comandada pelo processador, na etapa Pouso. Nesta fase, o processador envia uma mensagem `LAND` informando um *waypoint* de descida e o ângulo *heading* de aproximação. As transições desta fase vão dos estados *q5* ou *q7* e seguem em direção ao *q11*.

Ao ser enviada a ordem de pouso `LAND` e ser recebida da aeronave a mensagem `ACK` de confirmação, prossegue-se para o encerramento da missão e o fechamento da conexão. É a etapa de Encerramento. As transições de Encerramento, após uma missão ter sido bem sucedida, vão do estado *q12* para o estado final *q14*.

Para o caso em que, terminada a primeira fase, o processador definiu a inviabilidade da execução da missão, por graus de incompatibilidade entre processador e aeronave, a etapa seguinte deve ser a de Encerramento, ao invés de Execução. Neste caso, a transição partirá do estado *q2* rumo aos estados *q13* e *q14*.

Nota-se ainda, as transições `ABORT` em $q7 \rightarrow q14$ e $q8 \rightarrow q14$, que são disparadas pela aeronave quando ocorre alguma interrupção interna de maior preempção e que causa o abortamento da execução da missão, fechando-se a conexão entre as entidades. Estas transições podem ser disparadas em tempo de execução da missão, quando um comando terrestre assume o controle da aeronave, ou algum código pré-programado assume o controle da aeronave para direcioná-la para outra tarefa, como pouso forçado por fim de combustível, ou mudança de rota por invasão de território estrangeiro, etc. Esta primitiva do protocolo disponibiliza ao processador de missão a informação de que não possuirá mais comunicação com a aeronave; e disponibiliza também uma liberdade para as plataformas de avião não tripulado de assumir o próprio controle em situações que exijam isso.

O protocolo SSP foi modelado para a comunicação entre uma entidade de processamento de missão – neste trabalho, especificadamente a plataforma MOSA – e uma pla-

taforma de VANT, por meio de uma conexão única. Por este motivo, não foi modelado um esquema de endereçamento, nem acrescentados campos de endereços dentro das primitivas. Essa conexão de única via restringe a escalabilidade deste mecanismo, contudo, inerentemente fornece uma camada de segurança, que é uma característica mais interessante no cenário de sistemas embarcados críticos.

Como já escrito anteriormente, a mesma entidade nunca provoca duas transições de estados sucessivas. Esse característica do modelo de uso alternado do canal de comunicação é um mecanismo de alto nível para garantir a ordenação das mensagens. Apesar de não estarem presentes no modelo aqui apresentado, a garantia de integridade das mensagens pode ser acrescentada neste modelo facilmente, através de adição de um campo de verificação de integridade no pacote de mensagens.

Por estar modelado na camada de aplicação e ter como objetivo principal a interconexão entre duas plataformas distintas, o protocolo pode aproveitar o benefício da técnica de troca de mensagens, amplamente utilizada neste tipo de cenário (Hura e Singhal, 2001). Por assim ter sido feito, também não foi modelado um mecanismo de fragmentação de mensagens, deixando este sob responsabilidade da camada de transporte inferior.

4.4 Gramática do Protocolo

Para que a comunicação ocorra entre duas entidades, não basta definir somente o vocabulário que elas devem utilizar. É necessário estabelecer regras para o agrupamento dos termos do vocabulário. A modelagem do protocolo SSP também abrangeu a descrição de uma gramática para definir as regras de utilização das primitivas de serviço.

A Sintaxe das Mensagens

A sintaxe da gramática foi descrita formalmente utilizando o Formalismo de Backus-Naur (BNF – *Backus-Naur Form*), que é uma meta-sintaxe usada para expressar gramáticas livres de contexto [do inglês *Context-free Grammar (CFG)*]. As CFG são gramáticas formais em que cada regra de produção tem a notação $V \mapsto w$, onde V é um único símbolo não-terminal, e w é uma cadeia de terminais e/ou não-terminais ou a cadeia vazia.

A BNF é amplamente usada como uma notação para as gramáticas de linguagens de programação, de conjunto de instruções em projetos de compiladores e de protocolos de comunicação (Knuth, 2003), e por isso foi aplicada neste trabalho.

Uma especificação BNF é um conjunto de regras de produção ou derivação, escritas como a seguinte notação:

```
<símbolo> ::= <expressão com símbolos>
```

onde `<símbolo>` é um não-terminal, e `<expressão com símbolos>` consiste em sequências de símbolos separados pela barra vertical `|` , indicando uma escolha para as possibilidades de substituição do símbolo à esquerda do sinal `::=` . Os símbolos que nunca aparecem no lado esquerdo são ditos terminais. Na especificação feita neste trabalho, acrescentou-se também comentários sobre as regras de produção, que iniciam-se com o símbolo de porcentagem (`%`).

A descrição da gramática criada está anexada à esta dissertação, no Apêndice A e não está listada logo em sequência pois somente sua leitura não seria suficiente para a compreensão da comunicação sobre o SSP. Para isto, faz-se necessário apresentar também a semântica das mensagens, o que é feito na próxima seção.

A Semântica das Mensagens

As mensagens SSP possuem o mesmo tamanho de cabeçalho. A quantidade de bits de dados que elas carregam depende da mensagem a ser enviada.

Na gramática definida e listada no Apêndice A foram definidas duas unidades de medida: `<integer>` e `<decimal>`, ambas com 2 bytes (16 bits) de tamanho.

O cabeçalho SSP é definido por dois campos, que utilizam 48 bits de espaço na mensagem. O primeiro campo informa a versão (definida na gramática como “1.0”) do protocolo, com 3 caracteres, o que ocupa 3 bytes (ou 24 bits). O segundo campo informa a primitiva de serviço que esta mensagem carrega. Todas as primitivas SSP são formadas por 3 caracteres, ocupando os 24 bits restantes do cabeçalho da mensagem.

O formato das mensagens SSP é ilustrado na Figura 4.4.

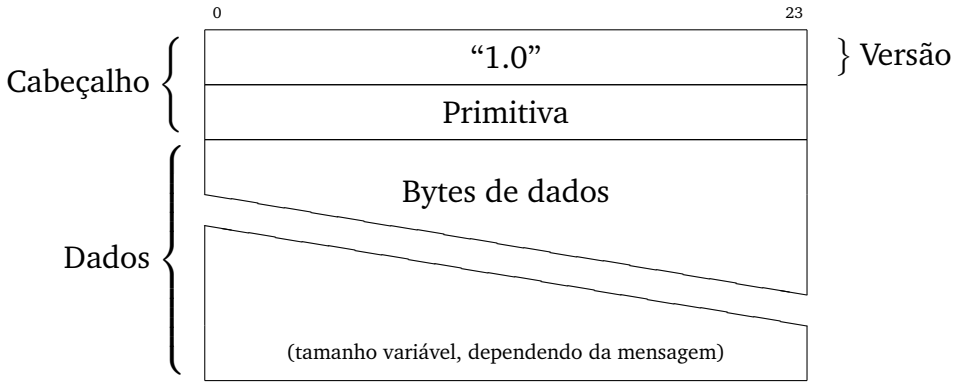
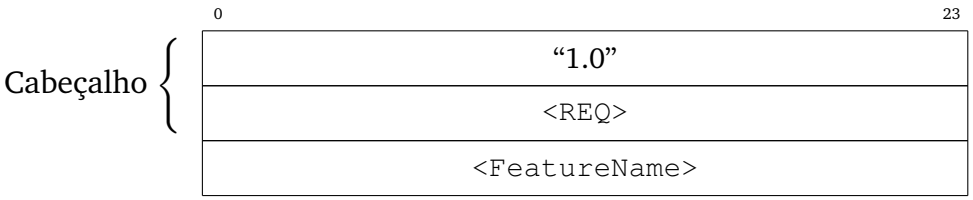


Figura 4.4: Formato de um pacote SSP

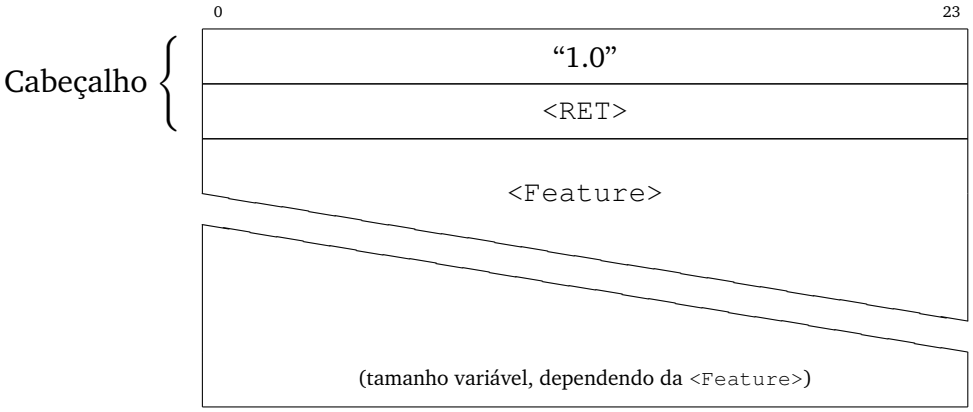
Etapa de Negociação

A etapa de Negociação serve para o processador de missão validar a aeronave (ou, até mesmo, o veículo autônomo seja ele aéreo, terrestre ou aquático/subaquático) à qual ele está conectado como um meio de transporte compatível com a missão a ser executada.

A primeira mensagem da Negociação é do tipo <REQ>, emitida pelo processador de missão para o VANT. Para cada mensagem de requisição a aeronave responde com uma mensagem do tipo <RET>. Os pacotes <REQ> e <RET> são ilustrados na Figura 4.5.



(a) Pacote <REQ>



(b) Pacote <RET>

Figura 4.5: Pacotes SSP da etapa de Negociação

A seguir são descritas as mensagens trocadas nesta etapa (Negociação). O campo <Versão> do protocolo foi omitido pois seria redundante apresentá-lo em todas as mensagens.

Parte A – Caracterização do veículo Nesta etapa são trocadas informações a respeito das características do veículo com relação ao seu ambiente de operação.

<REQUEST> | *Env*	o processador da missão requisita o tipo de ambiente que a aeronave é compatível. O intuito desta mensagem é identificar a plataforma do veículo não tripulado [*Unmanned Aerial Vehicle* (UAV), *Unmanned Ground Vehicle* (UGV), *Unmanned Surface Vehicle* (USV) ou *Unmanned Undersea Vehicle* (UUSV)] à qual se conectou, e é o primeiro parâmetro da comunicação a ser requisitado;

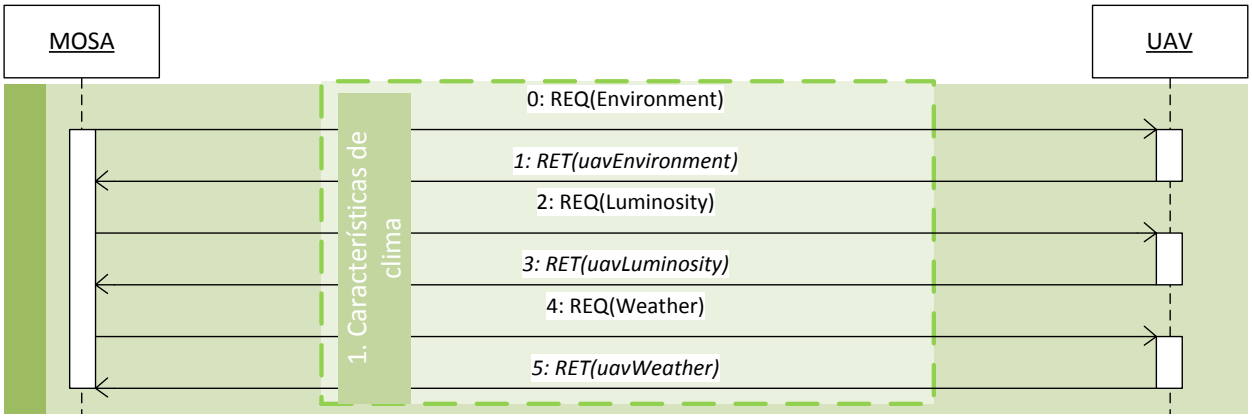


Figura 4.6: Sequência de mensagens para reconhecimento do veículo conectado ao MOSA (recorte da Figura 4.3)

`<RETURN> | <Environment>`	a aeronave responde com uma pacote do tipo `<Environment>`. No caso de VANTs, sempre será retornado *`Air`*, o que identifica o veículo como um do tipo aéreo;
`<REQUEST> | Lum`	o controle da missão requisita o horário de operação da aeronave;
`<RETURN> | <Luminosity>`	a aeronave responde um pacote do tipo `<Luminosity>`;
`<REQUEST> | Wea`	o controlador da missão requisita os tipos de tempo aos quais a aeronave é compatível, como por exemplo, ensolarado, seco, chuvoso, etc (ver Apêndice A);
`<RETURN> | <Weather>`	a aeronave responde um pacote do tipo `<Weather>`;

Parte B – Funcionalidades de aeronaves Nesta etapa são trocadas informações a respeito das características específicas das aeronaves, normalmente utilizadas nas descrições de uma missão.

`<REQUEST> | Alb`	o processador da missão requisita os limites de altitude mínimo e máximo que a aeronave é capaz de voar;
`<RETURN> | <AltitudeBoundaries>`	a aeronave responde com um pacote do tipo `<AltitudeBoundaries>`;

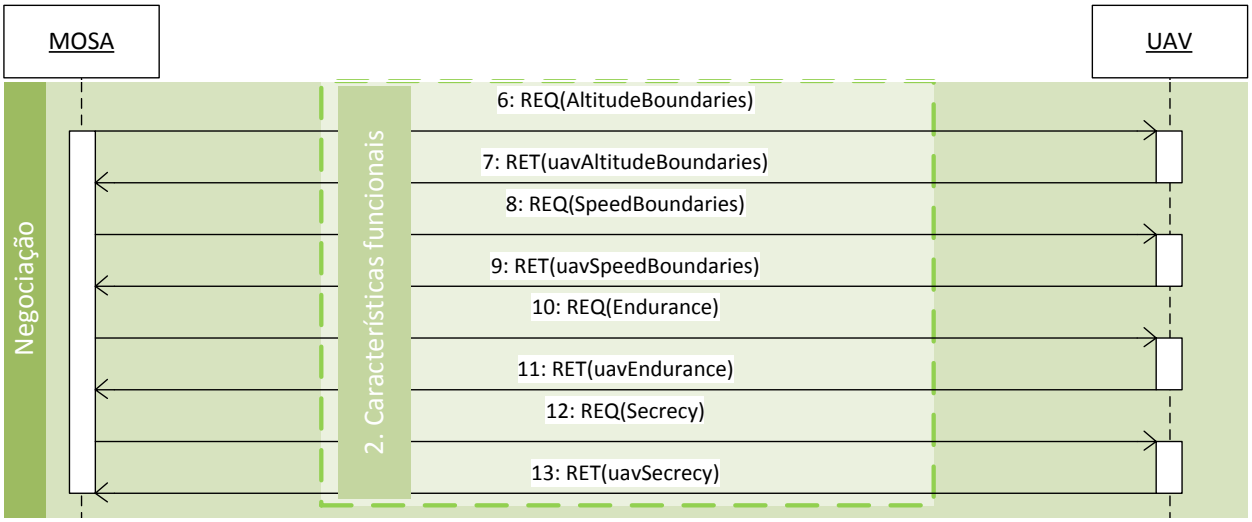


Figura 4.7: Sequência de mensagens para reconhecimento das características da aeronave conectada ao MOSA (recorte da Figura 4.3)

`<REQUEST> | Spb`	o processador da missão requisita os limites de velocidade de voo da aeronave;
`<RETURN> | <SpeedBoundaries>`	a aeronave responde com um pacote do tipo `<SpeedBoundaries>`;
`<REQUEST> | End`	o processador da missão requisita o quanto de tempo de voo a aeronave suporta;
`<RETURN> | <Endurance>`	a aeronave responde com um pacote do tipo `<Endurance>`;
`<REQUEST> | Sec`	o processador da missão requisita o tipo de operação para a qual esta aeronave foi projetada;
`<RETURN> | <Secrecy>`	a aeronave responde com um pacote do tipo `<Secrecy>`;

Parte C – Caracterização de arquitetura orientada a serviços embarcada Nesta etapa são trocados dados entre processos internos das plataformas, como, por exemplo, cliente e servidor de *webservices*.

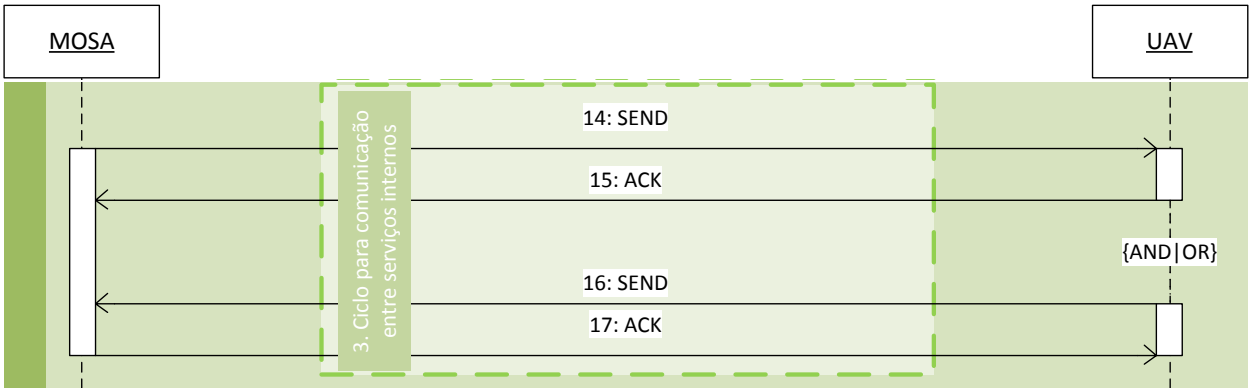


Figura 4.8: Ciclo de mensagens para reconhecimento da compatibilidade com a arquitetura SOA (recorte da Figura 4.3)

`<SEND> | <Lenght> | dados`	o processador da missão usa o pacote do tipo `SEND` para trocar dados entre clientes internos do processador e servidores internos da aeronave, como, por exemplo, no caso das operações de divulgação, descoberta e inscrição em serviços da arquitetura SOA. A aeronave também utiliza este tipo de pacote para enviar mensagens ou respostas do *webserver* para o cliente de *webservices* do processador de missão. Os dados da mensagem seguem encapsulados no campo *dados*, e no campo `<Lenght>` vai a quantidade de bytes dos dados;
`<ACK> | SND`	a aeronave e o processador de missão utilizam um pacote do tipo `<ACK>` para confirmar o recebimento de cada pacote `<SEND>`.

Um ciclo de troca de pacotes `<SEND` e `<ACK>` sempre deve ter um tempo finito de execução, pois a etapa de Execução da missão só pode ser iniciada depois deste ciclo.

Etapa de Execução

Ao terminar a etapa de Negociação o processador de missão deve ser capaz de definir a aeronave conectada como sendo compatível ou não à missão embarcada e a ser executada. Caso o VANT tenha sido avaliado como incompatível, a comunicação deve ser encerrada. Neste caso, uma mensagem solicitando o encerramento da missão deve ser enviada para a aeronave. Esta, por sua vez, deve informar o recebimento da mensagem. Os pacotes responsáveis por essas mensagens estão representados nas figuras 4.10(a) e 4.10(b), respectivamente.

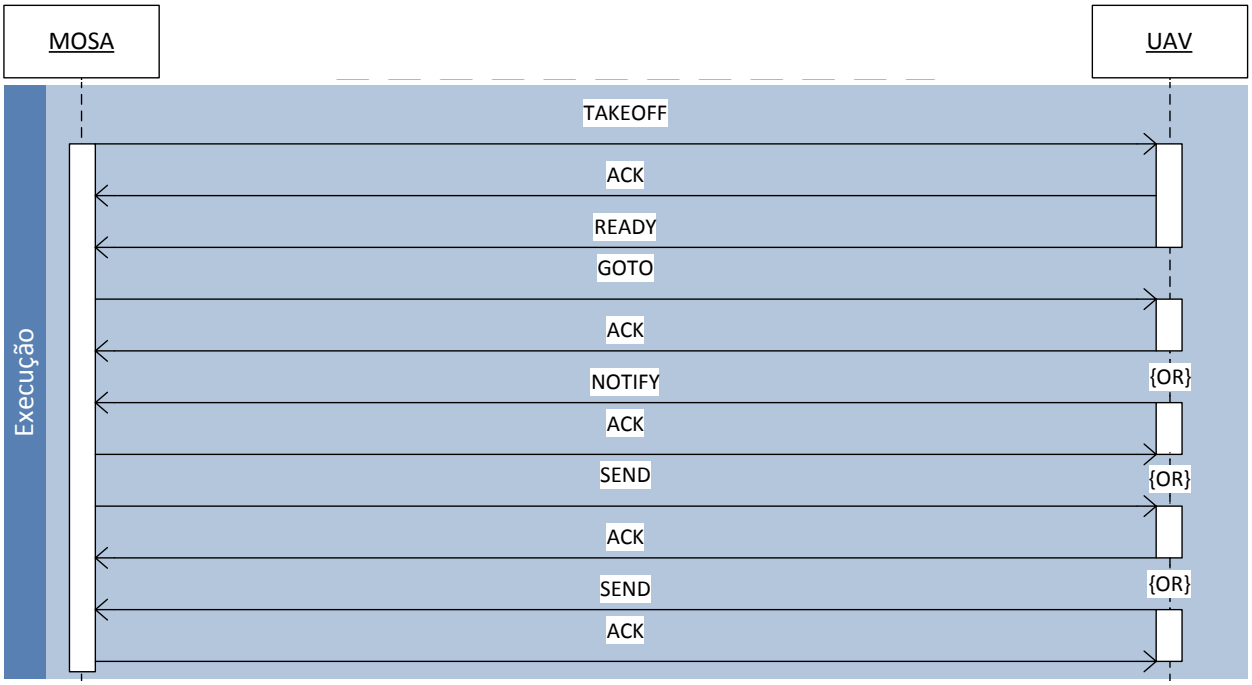
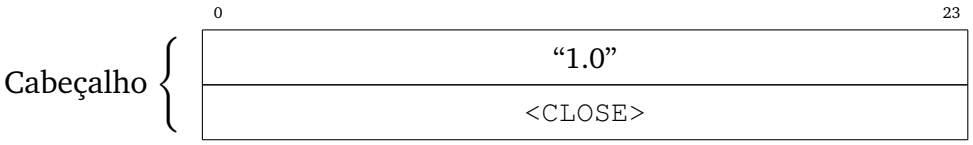
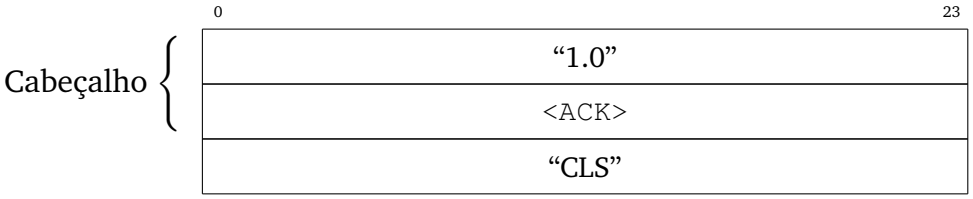


Figura 4.9: Ciclo de mensagens na Etapa de Execução da Missão (recorte da Figura 4.3)



(a) Pacote `<CLOSE>`

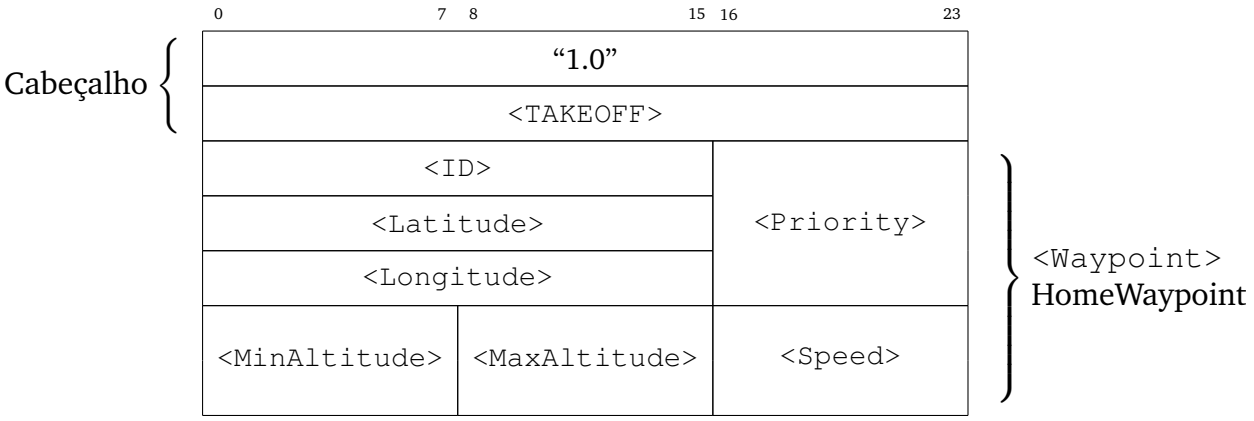


(b) Pacote `<ACK>` para o `<CLOSE>`

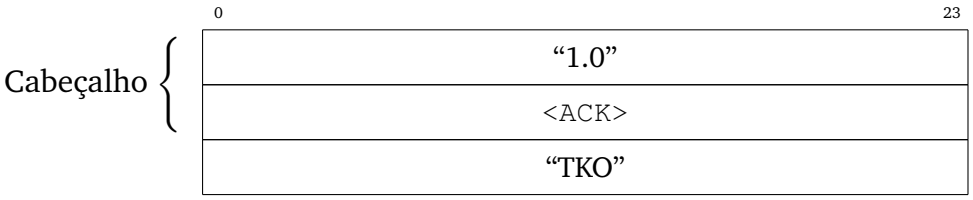
Figura 4.10: Pacotes SSP da etapa de Encerramento, usados para solicitar o encerramento da comunicação entre as entidades

Caso o VANT tenha sido avaliado como compatível com a missão, a comunicação continua ocorrendo e a cooperação passa para a próxima etapa: Execução da missão.

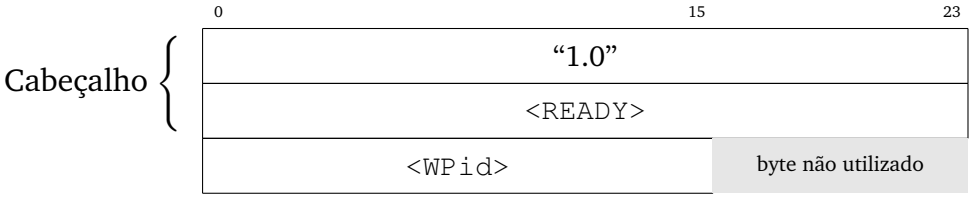
A etapa de Execução inicia-se com um pacote do tipo `<TAKEOFF>`, que é formado por 21 bytes, sendo 6 bytes do cabeçalho SSP e 15 bytes para os campos do `<HomeWaypoint>`. Este *waypoint* indicado o lugar onde a aeronave deve esperar pelo próximo comando, ou de onde ela deve começar a executar a missão. Assim que ela atinge a posição requisitada, a aeronave responde para o processador de missão uma mensagem `READY`. Ambos os pacotes estão representados nas figuras 4.11(a) e 4.11(c).



(a) Pacote <TAKEOFF>



(b) Pacote <ACK> para o <TAKEOFF>

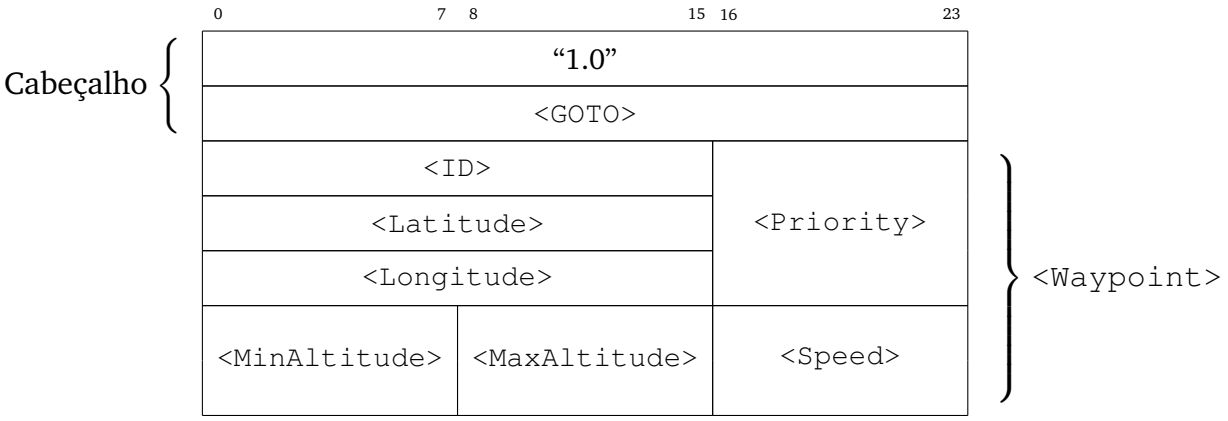


(c) Pacote <READY>

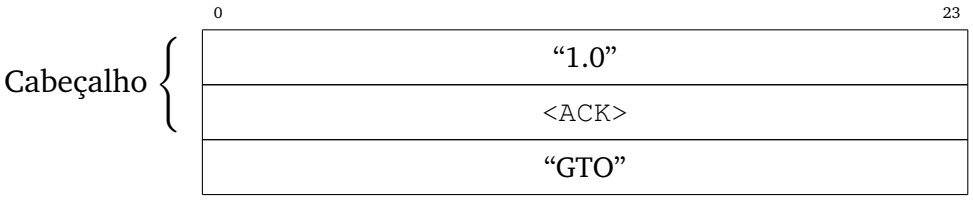
Figura 4.11: Primeiros pacotes SSP da etapa de Execução, usados para solicitar a decolagem da aeronave.

`<TAKEOFF> | <HomeWaypoint>` — o processador da missão solicita que a aeronave decole para o *waypoint* informado como argumento da mensagem;

`<ACK> | TKO` — a aeronave responde a solicitação com um pacote <ACK> informando que recebeu a solicitação. Em seguida o *middleware* da aeronave traduz essa requisição de deslocamento para a linguagem específica da plataforma da aeronave, para que seu sistema prossiga com a decolagem;



(a) Pacote <GOTO>



(b) Pacote <ACK> para o <GOTO>

Figura 4.12: Pacotes SSP da etapa de Execução, usados para solicitar um deslocamento da aeronave

<READY> | WPid

assim que a aeronave atinge o *waypoint* solicitado ela avisa o processador de missão através de uma mensagem <READY>.

Assim que a mensagem READY foi recebida e interpretada, a aeronave já pode ser deslocada. Uma requisição de deslocamento pode ser feita com uma mensagem <GOTO>, onde é informado como argumento um <Waypoint> indicando o ponto GPS para o qual a aeronave deve se mover. Toda mensagem <GOTO> que a aeronave recebe deve ser respondida com uma mensagem <ACK>. Estes pacotes estão representados nas figuras 4.12(a) e 4.12(b).

<GOTO> | <Waypoint>

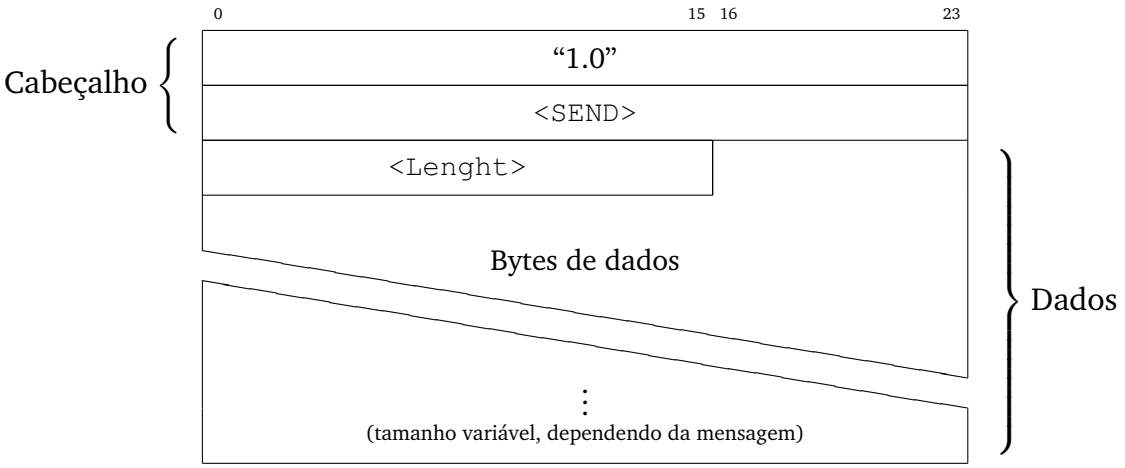
o processador da missão solicita que a aeronave se desloque para o próximo *waypoint* informado como argumento da mensagem;

<ACK> | GTO	a aeronave responde a solicitação com um pacote <ACK> informando que recebeu a solicitação. Em seguida o *middleware* da aeronave traduz essa requisição de deslocamento para a linguagem específica da plataforma da aeronave, para que seu sistema prossiga com o deslocamento;

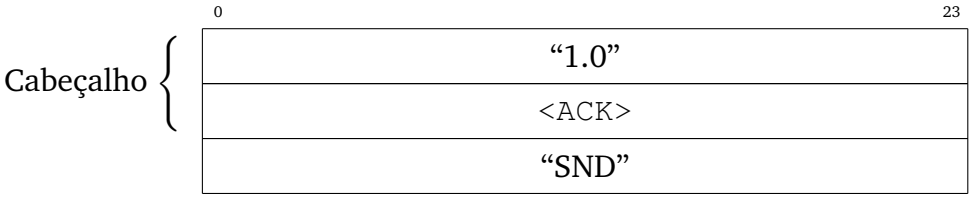
O barramento da conexão também é intercalado para ser utilizado com os pares de mensagens (<SEND>, <ACK>) e (<NOTIFY>, <ACK>).

O pacote <SEND> já foi explicado anteriormente, na fase de Negociação. Ele não possui tamanho fixo e, por isso, possui o campo <Length> para informar o tamanho dos dados que seguem encapsulados no campo seguinte. Este pacote é ilustrado pela Figura 4.13(a).

<NOTIIFY> | <WPid>	a aeronave notifica o processador de missão que uma localização solicitada foi alcançada. Como parâmetro o pacote carrega o <WPid> do <Waypoint> solicitado previamente. Este pacote está ilustrado na Figura 4.14(a).
<ACK> | NTF	Assim como acontece com o pacote <GOTO>, cada mensagem <SEND> e <NOTIFY> deve ser respondida com uma mensagem <ACK> confirmando a recepção da mensagem. Estas respostas estão representadas nos pacotes das Figuras 4.12(b), 4.13(b) e 4.14(b).

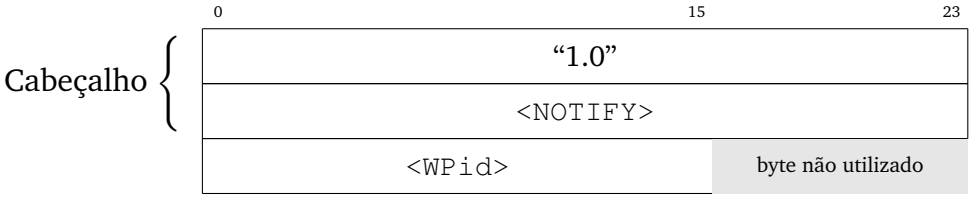


(a) Pacote <SEND>

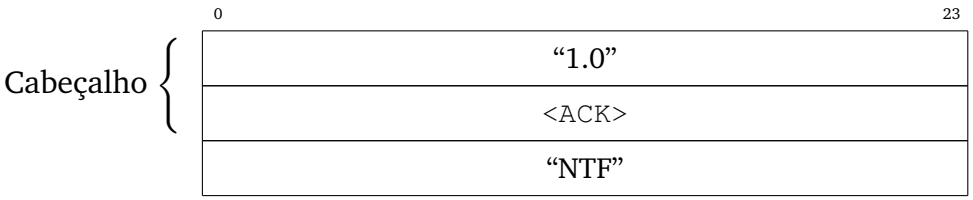


(b) Pacote <ACK> para o <SEND>

Figura 4.13: Pacotes SSP da etapa de Execução, usados para trocar dados diretamente entre processos internos da aeronave e do processador de missão



(a) Pacote <NOTIFY>



(b) Pacote <ACK> para o <NOTIFY>

Figura 4.14: Pacotes SSP da etapa de Execução, usados para notificar o alcance de um ponto GPS solicitado previamente

Etapa de Encerramento

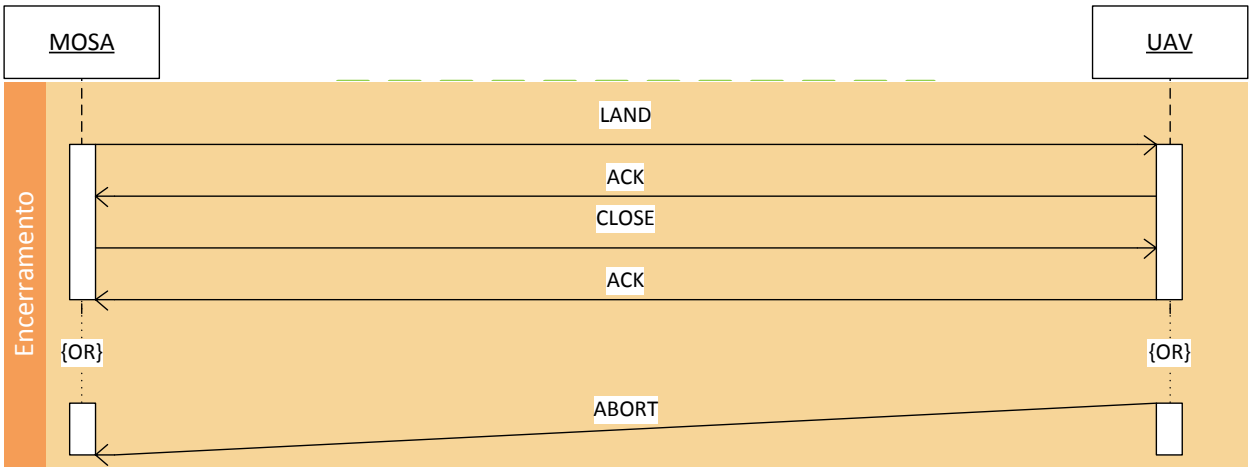
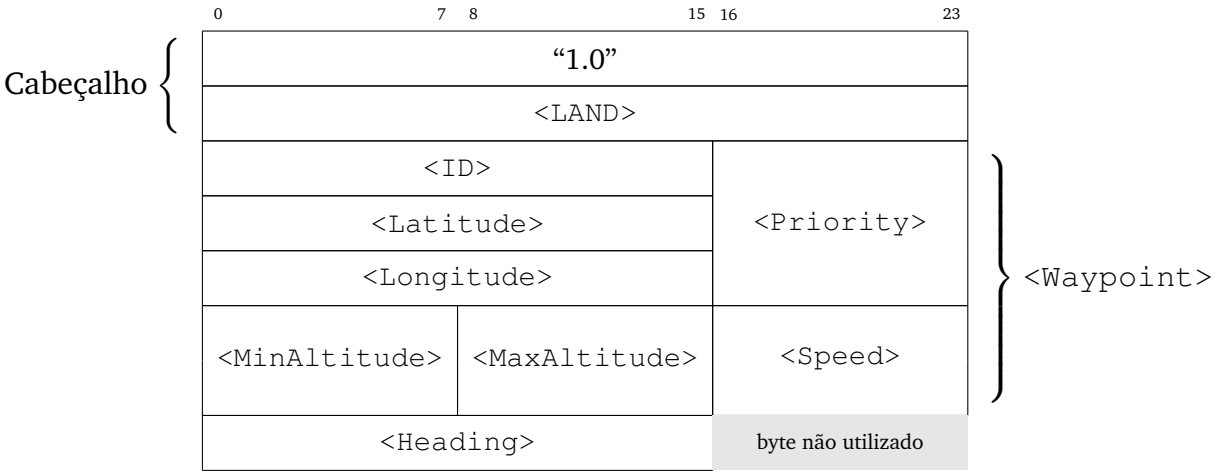


Figura 4.15: Ciclo de mensagens na Etapa de Execução da Missão (recorte da Figura 4.3)

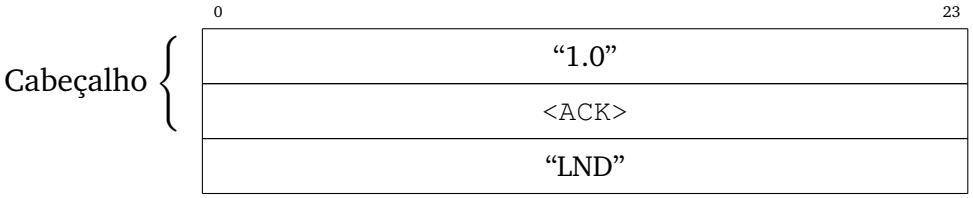
Ao terminar os pontos por onde a aeronave deve voar, o processador deve emitir uma mensagem de pouso para a aeronave, informando o ponto onde ela deve pousar e o ângulo de aproximação daquele ponto, informação esta que é importante para aeronaves de asa fixa, que precisam saber por onde deve ser feita a aproximação da cabeceira da pista, visto que não conseguem pousar na vertical como aeronaves de asas rotativas. Em seguida a comunicação deve ser encerrada com uma mensagem `<CLOSE>` (Figura 4.10(a)).

A Figura 4.16(a) apresenta o modelo do pacote `<LAND>`.

Durante a execução da missão, alguma interrupção de maior preempção que a execução da missão pode ocorrer, como por exemplo, o comando terrestre pode assumir o controle da aeronave, ou o nível de combustível torna-se muito baixo e o sistema de controle da aeronave decide interromper a missão e voltar ao ponto de partida, ou, ainda, o sistema de ciência de aeronavegabilidade percebe que a aeronave sobrevoa um território não permitido e decide retornar etc. Ao ocorrer isto, o sistema de controle assume completamente o controle de voo da aeronave e pára de aceitar novas requisições. Ao mesmo tempo é necessário informar ao processador de missões que a cooperação está sendo abortada e que a comunicação será encerrada abruptamente logo após o aviso. Este tipo de assunção permite ao sistema de controle da aeronave o controle soberano sobre si, o que é necessário em sistemas que precisam prezar por sua integridade.



(a) Pacote <LAND>



(b) Pacote <ACK> para o <LAND>

Figura 4.16: Pacotes SSP da etapa de Encerramento, usados para proceder com o pouso da aeronave

<ABORT> | <Lenght> | *mensagem*

a aeronave notifica o processador de missão que a comunicação foi abortada e que não receberá nem responderá novas mensagens. Como parâmetro o pacote carrega a quantidade de bytes que seguem no próximo campo, que, por sua vez, pode ou não conter uma mensagem informando a causa do abortamento, para fins de *logging*. Este pacote está ilustrado na Figura 4.17.

Por representar uma retomada de controle de maior preempção, o emissor da mensagem <ABORT> não espera um <ACK> como resposta, por isso este pacote não está modelado.

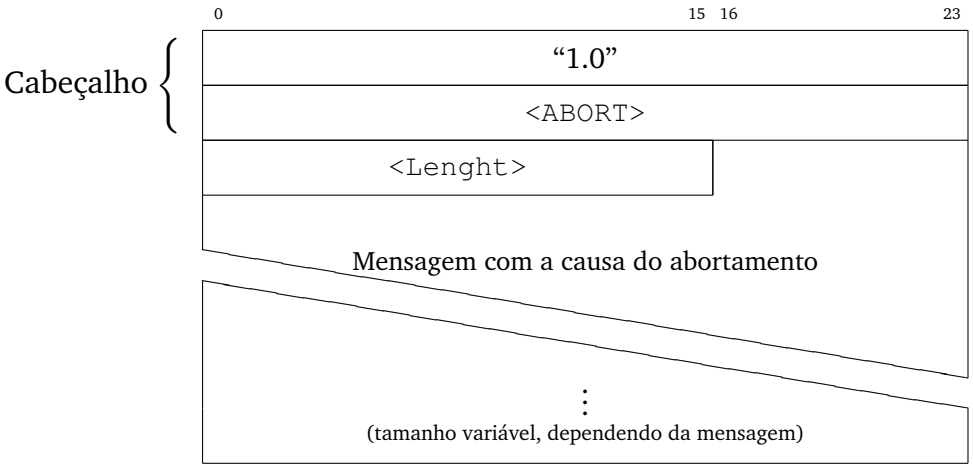


Figura 4.17: Pacote <ABORT>, utilizado para abortar a cooperação e encerrar a comunicação entre os *middlewares* da aeronave e do processador de missão

4.5 Considerações Finais

Neste capítulo foi apresentado todo o projeto do protocolo SSP, desde a etapa de sua especificação, utilizando textos e figuras para diagramar a estrutura das mensagens, passando pela arquitetura da comunicação e seguindo até a explicação da gramática criada para as primitivas desse protocolo.

CAPÍTULO

5

Experimentos Realizados

Para validar todo o projeto realizado e descrito no Capítulo 4 fez-se tanto a validação formal do modelo, quanto a implementação em uma plataforma de hardware para averiguar sua funcionalidade em um ambiente de simulação. Sendo assim, os experimentos realizados são apresentados neste capítulo em duas partes.

5.1 Validação do modelo

Para verificar e testar o protocolo desenvolvido, esse sistema foi modelado em uma ferramenta de validação e teste de modelos já utilizada com protocolos de comunicação consolidados. A verificação de modelos é uma técnica poderosa para se fazer depuração de sistemas reativos complexos, como é o caso de protocolos de comunicação. Na verificação de modelos, as especificações de um sistema são feitas com fórmulas lógicas e algoritmos simbólicos eficientes são utilizados para analisar o modelo e cruzá-lo com aquela especificação. As ferramentas de verificação de modelos capturam o comportamento dinâmico dos sistemas utilizando para isto apenas estados e transições entre estados do modelo.

A ferramenta utilizada para fazer a verificação do protocolo SSP foi a UPPAAL[1], na versão 4.0.13. Como descrita pelos próprios autores em (Behrmann et al., 2004), a UPPAAL é um conjunto de ferramentas para a tarefa de verificação de sistemas de tempo real, desenvolvida em conjunto pela Uppsala University, da Suécia, e pela Aalborg University, da Dinamarca.

[1] `http://www.uppaal.org/`

Dentre as ferramentas encontradas para fazer este tipo de tarefa, a UPPAAL foi escolhida pelos diversos estudos de casos de sucesso de modelagem de protocolos de comunicação já estabelecidos e amplamente utilizados comercialmente (Ravn et al., 2010b,a; Hessel e Pettersson, 2006; Bordbar et al., 2005; Bengtsson et al., 2002; Lindahl et al., 2001; David e Yi, 2000; Lindahl et al., 1998; Lönn e Pettersson, 1997; Havelund et al., 1997).

Na UPPAAL os componentes do sistema são chamados de *templates*. Os dois componentes principais do modelo implementado são o MOSA e o VANT.

O *template* MOSA é o componente que representa o processador de missão, que possui um único canal de comunicação com o *template* VANT, que como seu próprio nome indica, representa a aeronave não tripulada. O MOSA e o VANT foram modelados como ilustrado nas figuras 5.1 e 5.2, respectivamente.

Para aproximar ao máximo o experimento do ambiente real para o qual o protocolo foi modelado e também facilitar a visualização da execução do sistema, as funcionalidades da aeronave foram divididas em três componentes – VANT_negotiator, VANT_wpProcessor e VANT_msgProcessor – os quais possuem um canal de comunicação exclusivo com o *template* VANT.

O *template* VANT_negotiator (Figura 5.3) é responsável apenas por receber as requisições da etapa de Negociação e respondê-las. O *template* VANT_wpProcessor (Figura 5.4) é responsável por receber os pacotes `<GOTO>`, atendê-los e notificar o MOSA quando os *waypoints* forem alcançados. O *template* VANT_msgProcessor (Figura 5.5) é responsável por receber, encaminhar e responder os pacotes `<SEND>`. Ele funciona como um *gateway* entre o cliente e o servidor de *webservices*.

Todos estes *templates* juntos formam o sistema que o simulador do programa executa. O simulador inicia todos os *templates* nos seus estados iniciais e faz a transição dos estados, alimentando como entrada um dos possíveis parâmetros definidos. Sendo assim, a execução é não-determinística, mas o sistema é verificado, também, quanto ao seu determinismo. O simulador foi executado diversas vezes para que fossem visualizadas as mudanças sincronizadas dos vários componentes do sistema, bem como identificar todas as partes onde a execução era interrompida. Ao ser interrompido, verificava-se o estado de cada componente, o que ajudou a gerar mudanças incrementais no modelo, por causa da identificação de problemas.

O simulador oferece a funcionalidade de visualização, em tempo real, dos estados do sistema durante sua execução mediante entradas não-determinísticas. Essa funcionalidade viabiliza a verificação do sistema e sua validação perante a sua especificação. A fase de teste do protocolo é feita utilizando-se regras da Lógica Temporal.

O termo Lógica Temporal é usado para descrever qualquer sistema de regras e simbolismo para representar proposições qualificadas em termos do tempo. Qualquer lógica

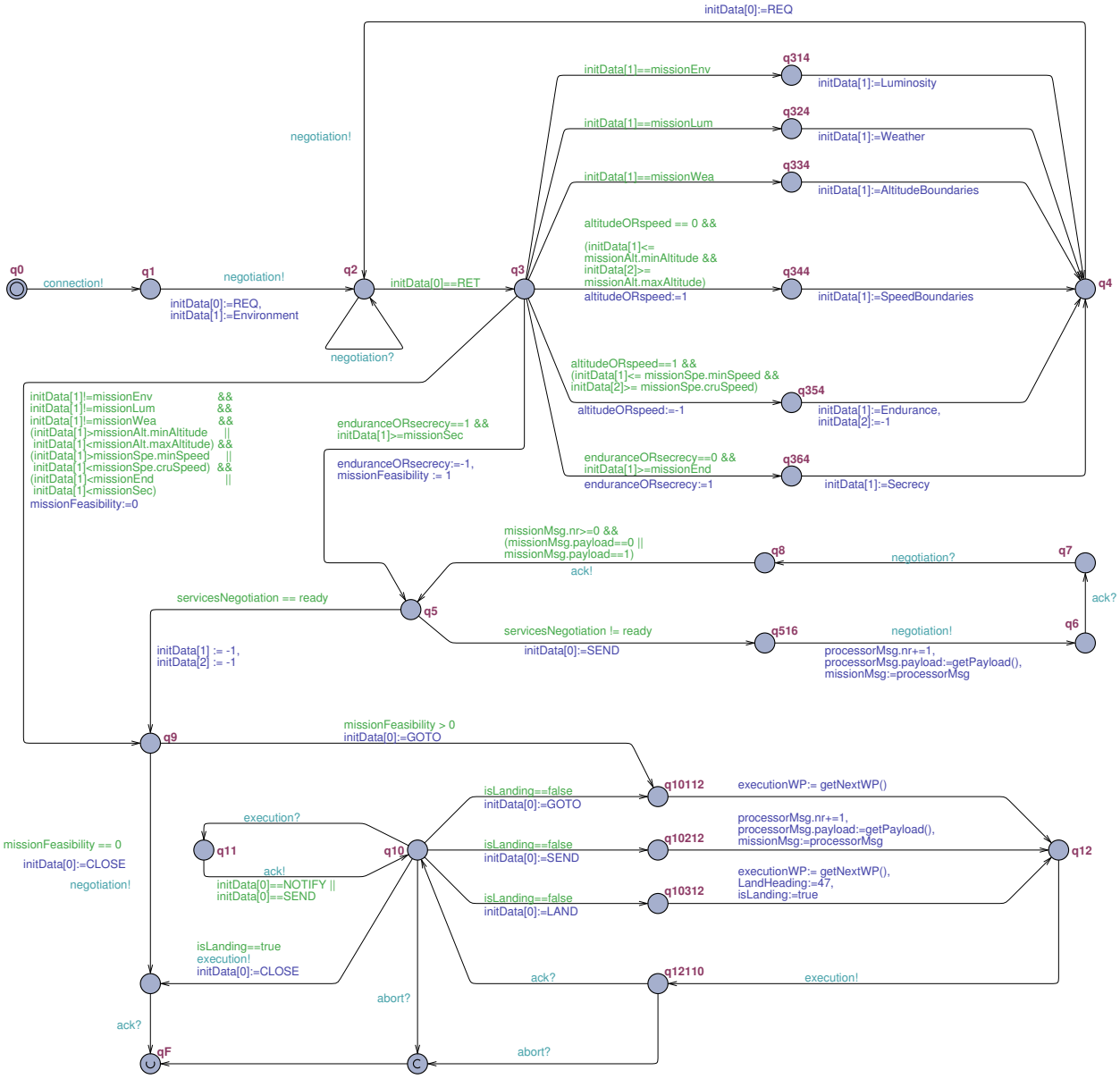


Figura 5.1: Comportamento do *middleware* do processador de missão, implementado na ferramenta UPPAAL

que utilize apenas dois valores para verdade é uma lógica binária, e qualquer lógica que considera o tempo como uma sequência de estados é uma lógica temporal (Galton, 2008).

A Lógica Temporal tem uma aplicação importante na verificação formal para estabelecer requisitos de *hardware* e *software*. Por exemplo, pode-se desejar que *em algum momento* uma requisição seja feita, ou que o acesso a um recurso seja *eventualmente* garantido, mas ele *nunca* será garantido a dois requisitantes simultaneamente. Este exemplo é um exemplo claro para ser expressado em Lógica Temporal. E é assim que o protocolo SSP foi expressado na parte de testes da ferramenta UPPAAL (Galton, 2008).

A ferramenta reconhece as propriedades *Possivelmente*, *Invariavelmente*, *Provavelmente sempre*, *Eventualmente* e *Conduz a*, que são descritas a seguir.

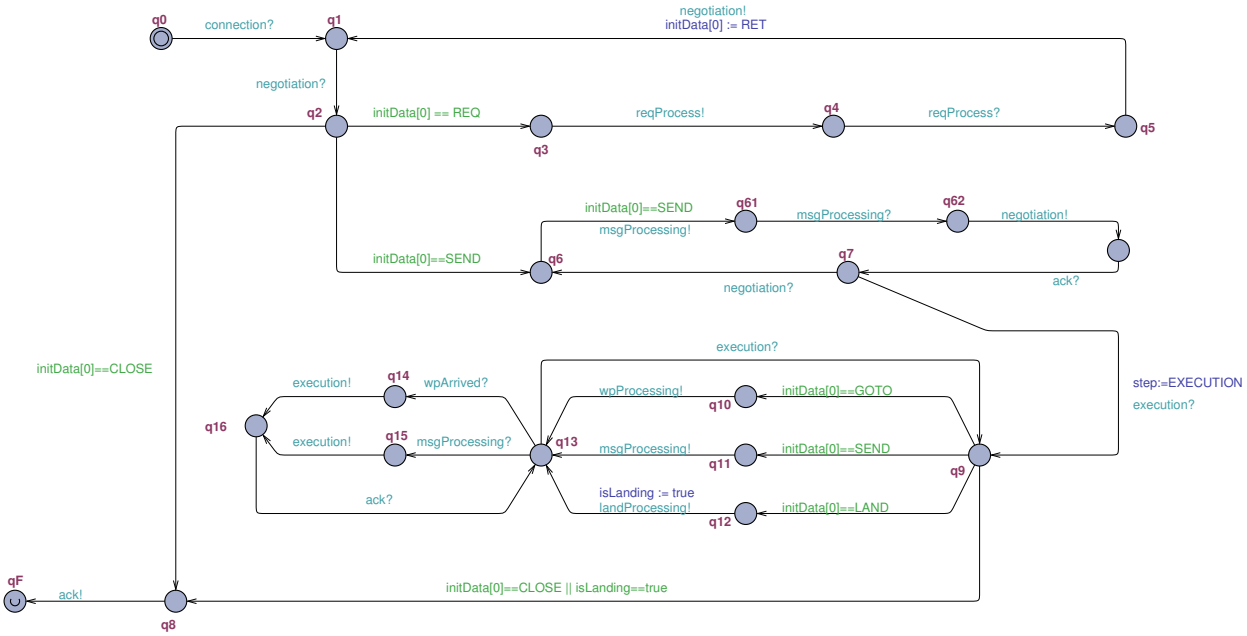


Figura 5.2: Comportamento do *middleware* da aeronave, implementado na ferramenta UPPAAL

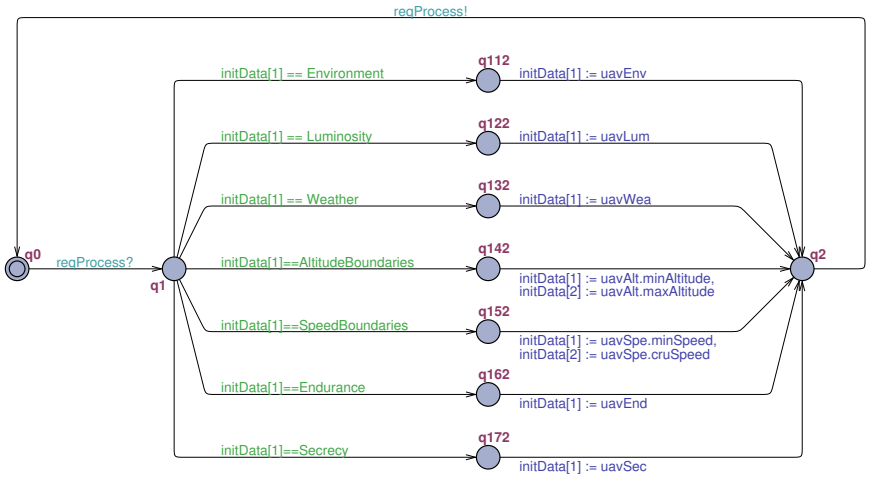


Figura 5.3: Comportamento do *middleware* da aeronave ao processar requisições <REQ>, implementado na ferramenta UPPAAL

Sejam p e q propriedades de estados para as cincos propriedades temporais supracitadas.

A propriedade $E <> p$ (Possivelmente) avalia para verdadeiro um sistema de transição temporizada se, e somente se, existe uma sequência alternante de transições de atraso e transições de ação $s0 \rightarrow s1 \rightarrow \cdots \rightarrow sn$, onde $s0$ é o estado inicial e sn satisfaz p.

A propriedade $A[]p$ (Invariavelmente) avalia para verdadeiro se, e somente se, todo estado alcançável satisfaz p. Uma propriedade $A[]p$ pode ser expressa como $not\ E <>\ not\ p$.

A propriedade $E[]p$ (Provavelmente sempre) avalia para verdadeiro um sistema de transição temporizada se, e somente se, existe uma sequência alternante de transições

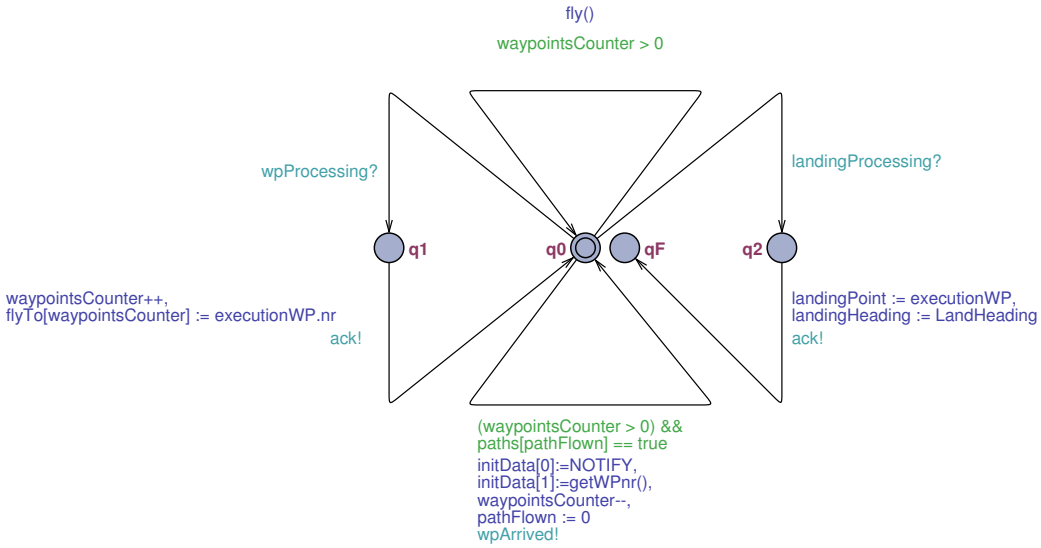


Figura 5.4: Comportamento do *middleware* da aeronave ao processar requisições <GOTO>, implementado na ferramenta UPPAAL

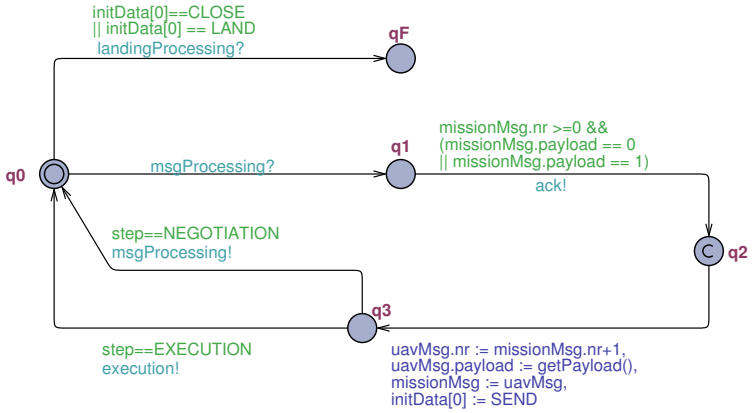


Figura 5.5: Comportamento do *middleware* da aeronave ao processar mensagens <SEND>, implementado na ferramenta UPPAAL

de atraso e transições de ação $s0 \rightarrow s1 \rightarrow \cdots \rightarrow si \rightarrow \ldots$, para a qual p é satisfeito em todos os estados si, e cada um deles:

- ou é infinito,
- ou termina em um estado (Ln, vn) tal que:
  - ou para todo $d : (Ln, vn + d)$ satisfaz p e $Inv(Ln)$,
  - ou não existe transição de saída de (Ln, vn)

A propriedade $A <> p$ (Eventualmente) avalia para verdadeiro se, e somente se, todas as possíveis sequências de transições eventualmente alcançam um estado que satisfaz p. Uma propriedade $A <> p$ pode ser expressa como $not\ E[]\ not\ p$.

A sintaxe $p \rightarrow q$ (Conduz à) significa que no momento em que p for satisfeita, q também o será. Uma propriedade $p \rightarrow q$ pode ser expressa com a propriedade $A[](p\ imply\ A <> q)$.

A UPPAAL também possui uma notação para representar as localizações de estados. Expressões na forma $P.l$, onde P é um processo e l é um local, avaliam para verdadeiro em um estado (L, v) se, e somente se, $P.l$ está em L.

Outra notação que é importante ressaltar neste ponto é a de *deadlock*. Diferentemente do conceito de Impasse no contexto dos sistemas operacionais, aqui a propriedade *deadlock* avalia para verdadeiro um estado (L, v) se, e somente se, para todo $d >= 0$ não existe uma ação sucessora de $(L, v + d)$.

As regras para o modelo criado na ferramente definem, essencialmente, que todos os processos devem começar no estado inicial e, quando algum dos processos alcançar o estado final, todos os outros processos também necessariamente devem ter alcançado seus respectivos estados finais.

Com base nos diagramas das figuras 5.1, 5.2, 5.3, 5.4 e 5.5 foram criadas as seguintes regras:

`E<> MOSA.q0`	Existe um caminho para o estado inicial do *template* MOSA;
`E<> VANT.q0`	Existe um caminho para o estado inicial do *template* VANT;
`E<> VANT_negotiator.q0`	Existe um caminho para o estado inicial do *template* VANT_negotiator;
`E<> VANT_msgProcessor.q0`	Existe um caminho para o estado inicial do *template* VANT_msgProcessor;
`E<> VANT_wpProcessor.q0`	Existe um caminho para o estado inicial do *template* VANT_wpProcessor;
`A[] MOSA.q0 imply VANT.q0`	Invariavelmente, o sistema inicia com os dois *templates* primários no estado inicial;
`A[] VANT.q0 imply (VANT_msgProcessor.q0 && VANT_wpProcessor.q0 && VANT_negotiator.q0)`	Invariavelmente o estado inicial do *template* VANT implica no estado inicial dos *templates* VANT_msgProcessor, VANT_wpProcessor e VANT_negotiator;
`A[] MOSA.q0 imply (not VANT.qF)`	Invariavelmente, quando o MOSA inicia o VANT não está no estado final;
`A[] (not VANT_msgProcessor.q0 || not VANT_wpProcessor.q0) imply (VANT_negotiator.q0)`	Invariavelmente, se a aeronave está processando uma mensagem ou um waypoint então a missão já deve ter sido negociada;

`E<> MOSA.qF`	Existe um caminho para o estado final para o *template* MOSA;
`E<> VANT.qF`	Existe um caminho para o estado final para o *template* VANT;
`A[] MOSA.qF imply VANT.qF`	Invariavelmente, o sistema termina (deadlock) com os dois processos no estado final;
`A[] (not MOSA.qF) imply (not VANT.qF)`	Invariavelmente, se o MOSA não está no estado final, então, o VANT também não está no seu estado final;
`A[] (not VANT_msgProcessor.q0 || not VANT_wpProcessor.q0) imply (VANT_negotiator.q0)`	Invariavelmente, se a aeronave está processando uma mensagem ou um waypoint então a missão já deve ter sido negociada;
`A[] MOSA.q0 imply (not VANT.qF)`	Invariavelmente, quando o MOSA inicia o VANT não está no estado final.

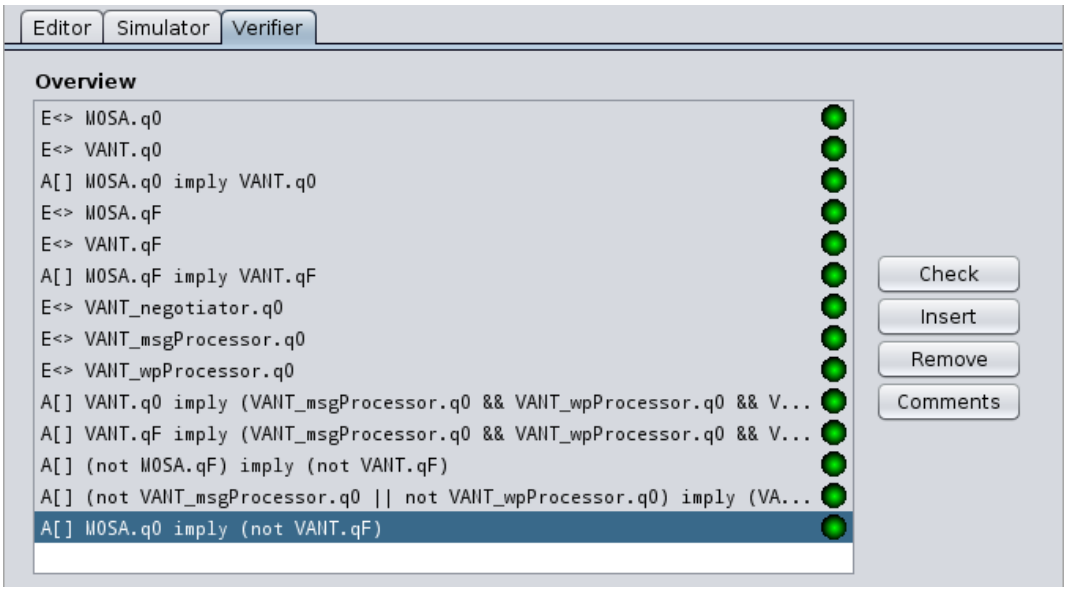
Todas as propriedades listadas anteriormente foram executadas pela UPPAAL e todas foram satisfeitas, como atestado na Figura 5.6, o que valida matematicamente o modelo criado.

5.2 Verificação da implementação

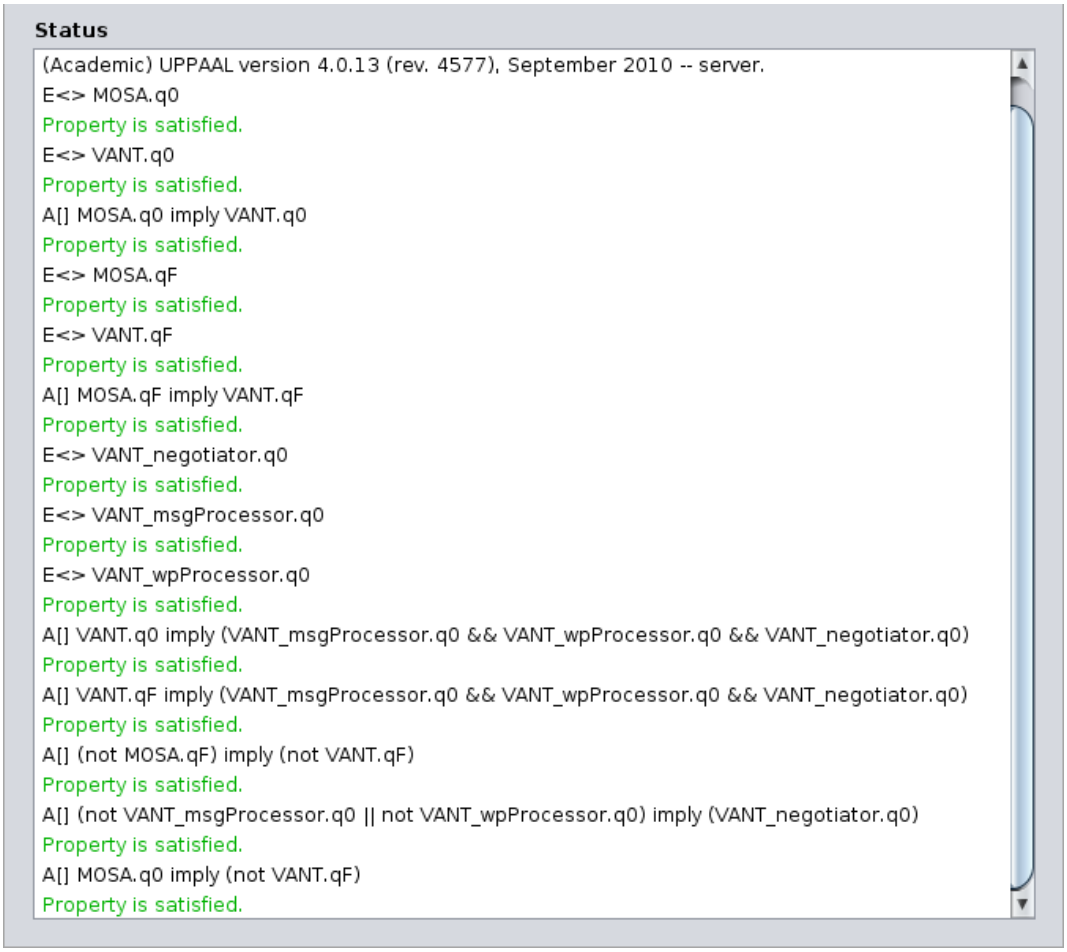
Para experimentar o modelo desenvolvido foi necessário criar uma missão para executá-la na ferramenta UPPAAL. Sendo assim, foi gerada uma gramática para uma missão e também uma missão de exemplo. A missão implementada na UPPAAL também foi convertida para um arquivo XML, seguindo a gramática, e ambos trabalhos estão listados nos apêndices B e C.

Nesse experimento, o protocolo SSP foi implementado em duas placas Arduino Mega 2560, a fim de se ter um protótipo de *hardware* já operando esse protocolo. Essas placas simularam o *middleware* SSP a ser utilizado para interfaceamento entre as entidades da comunicação.

Arduino é uma plataforma de prototipagem eletrônica de hardware livre, projetada com um microcontrolador Atmel AVR de placa única, com suporte de entrada/saída embutido, uma linguagem de programação padrão baseada na linguagem Processing, e é essencialmente C/C++. Uma placa Arduino típica é composta por um controlador, algumas linhas de E/S digital e analógica, além de uma interface serial ou USB, para comunicação com o computador, que é usado para programá-la e com ela interagir em tempo real.



(a) Regras testadas



(b) Resultado da verificação com cada regra

Figura 5.6: Resultado da verificação do modelo com regras da lógica temporal

A Arduino Mega 2560 é uma placa baseada no microcontrolador ATmega2560 (Arduino SA, 2013), com 54 pinos digitais de E/S [dos quais 15 podem ser usados como saídas PWM (*Pulse-Width Modulation*)], 16 entradas analógicas, 4 UARTs (porta em hardware serial), um cristal oscilador de 16MHz, uma conexão USB, um conector de energia, um conector ICSP (*In-Circuit Serial Programming*), e um botão de *reset*.

Além da Arduino[2], havia disponível comercialmente as placas BeagleBone[3], Raspberry Pi[4], Nanode[5] e Libelium Waspmote[6]. A BeagleBone e a Raspberry Pi necessitavam de um sistema operacional para operar, o que não fazia parte do escopo do protocolo SSP. A Waspmote foi desenvolvida para um ambiente de operação sem-fios, e o cenário de operação do SSP no abrange este tipo de conexão. A Nanode é muitíssimo parecida com a Arduino, contudo, a facilidade na aquisição dessa última e a disponibilidade de materiais para consulta, inclusive na própria biblioteca do Instituto (ICMC) destacaram a Arduino Mega 2560, dentre todas as possibilidades, como a opção mais viável para os experimentos desejados. Ainda, ela atende à capacidade de se ter duas portas seriais sendo utilizadas concomitantemente, pois havia tráfego entre duas placas Arduino e entre cada placa e seu computador.

Para representar o cenário apresentado na Figura 4.1, montou-se o seguinte cenário para o experimento: Assumiu-se um computador como o processador de missão e outro computador como a aeronave não tripulada. A comunicação entre ambos computadores ocorreu por meio das duas placas Arduino do modelo supracitado, onde estava implementado o SSP. Os computadores utilizaram uma conexão USB para se comunicar com sua respectiva placa arduino, e as placas arduino comunicaram-se utilizando apenas um canal de entrada/saída serial, a uma velocidade de 115200bps.

O ambiente da simulação é ilustrado na Figura 5.7, e uma fotografia das placas montadas no experimento é ilustrada na Figura 5.8.

Neste experimento a execução de uma missão foi simulada, fazendo com que a troca de mensagens entre as entidades passasse pelos *middlewares*. A missão codificada está listada no Apêndice C.

Este experimento ajudou a reforçar a ideia de que não é necessário a utilização de uma estrutura de hardware sofisticada para a implementação do modelo proposto neste trabalho. Um microcontrolador simples, com um *clock* de 16MHz e 256KB de memória flash foi capaz de rodar o protocolo proposto, o que viabiliza muito a sua implantação em cenários reais e a difusão deste tipo de veículo.

[2] `http://www.arduino.cc/`
[3] `http://beagleboard.org/`
[4] `http://www.raspberrypi.org/`
[5] `http://www.nanode.eu/`
[6] `http://www.libelium.com/`

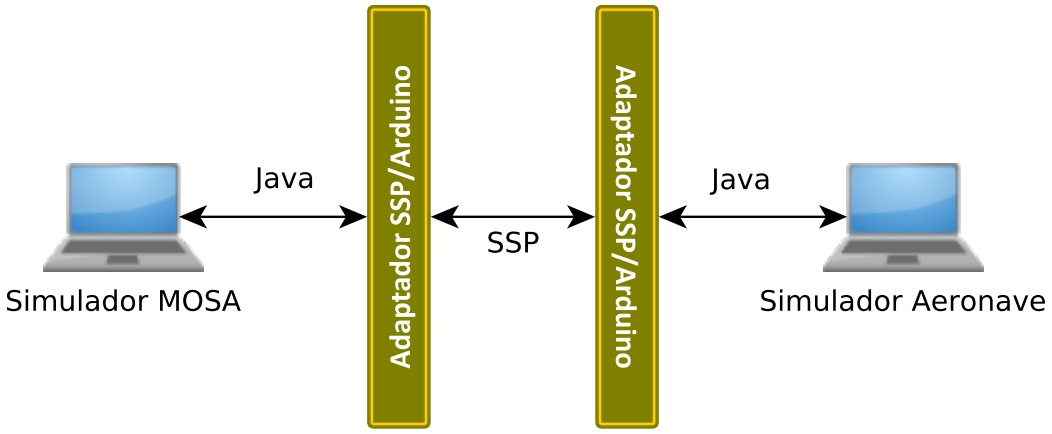


Figura 5.7: Representação do ambiente de simulação

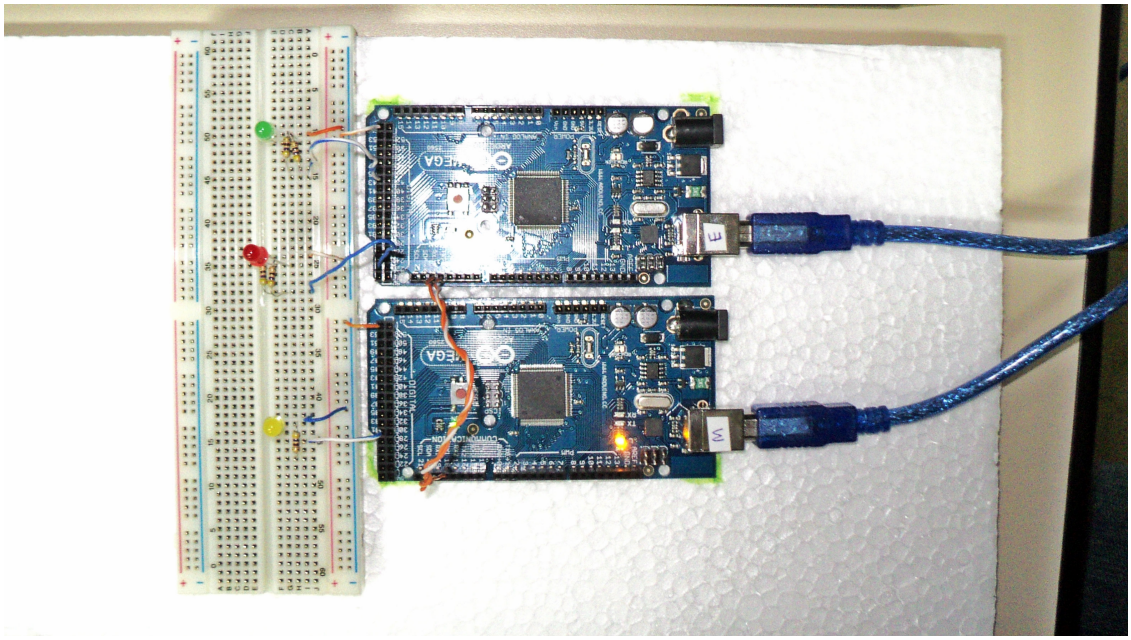


Figura 5.8: Arduinos utilizados no experimento

5.3 Considerações Finais

Este capítulo apresentou todos as etapas de todos os experimentos realizados neste trabalho. Começou pela validação do modelo criado e explicado no capítulo anterior, feita por meio da ferramenta UPPAAL e sua linguagem de programação própria. Depois apresentou a verificação matemática e lógica do modelo, feita também na mesma ferramenta, através de regras da Lógica Temporal. Por fim, apresentou a implementação do protocolo em hardware de prototipação Arduino e o ambiente de simulação de uma missão executada com o apoio do SSP.

CAPÍTULO

6

Conclusão

Veículo aéreos não tripulados são bastante empregados em operações militares desde a década de 60, como no caso dos ataques do Vietnã à extinta União Soviética. Recentemente, são protagonistas em diversas reportagens em que citam o seu uso, principalmente, pelos exércitos dos Estados Unidos e de Israel. Mas, além do uso militar, têm-se visto crescer e aparecer diversos usos em aplicações civis, como o uso pela polícia civil para fiscalização de operações de empresas que exploram o meio ambiente, a utilização por empresas do ramo agroindustrial para o reconhecimento de falhas e detecção de doenças em plantações, e também o controle de multidão em eventos como a copa do mundo, além de vários empregos na indústria cinematográfica.

Essas crescentes e evidentes ampliação nas possibilidades de emprego e demanda pela utilização destes robôs acarretam em um aumento também em sua produção, na facilidade de acesso e na redução do custo. Contudo, essa grande variedade de aplicações para esses robôs acabam deixando bastante heterogêneas a programação dessas aeronaves.

Nesse contexto, a oportunidade de criar um mecanismo de inteligência para o acoplamento entre diversas plataformas de processamento de missões com os diferentes VANTs veio à tona.

A arquitetura que aqui foi proposta tem como objetivo desassociar do sistema de controle da aeronave a responsabilidade de gerenciar e executar as tarefas programadas na missão. Desta forma, a aeronave trabalha somente como um veículo de transporte para o processador de missões, recebendo e atendendo às suas solicitações de deslocamento. Este processador inteligente de missões também foi proposto em um trabalho derivado desta dissertação, publicado em (Pires et al., 2011).

Essa separação de responsabilidades entre dispositivos diferentes também deve contribuir para o processo de certificação das mesmas, pois a certificação dessas aeronaves não tripuladas (ou de qualquer veículo não tripulado) pode ser feita independentemente da *payload* que ela vai transportar, ficando a cargo do responsável pelo desenvolvimento do MOSA a certificação de cada arranjo. Essa contribuição com o processo de certificação dessas aeronaves deverá facilitar também a inserção dos VANTs em um espaço aéreo compartilhado com as aeronaves tripuladas.

Os experimentos executados neste trabalho e apresentados no capítulo anterior mostram a viabilidade da implementação de um mecanismo de interfaceamento para o acoplamento de processadores inteligentes de missão em aeronaves não tripuladas, tanto para aplicações militares quanto para civis.

Para desenvolver o mecanismo aqui proposto foram levantadas técnicas de projeto de protocolos relevantes e de amplo uso na comunidade científica (ISO, 1989; Tarnay, 1991; Moecke e Farines, 1992; Turner, 1993; ISO, 1997; ITU, 1999; Cordeiro et al., 2003; Behrmann et al., 2004; Ferreira, 2005; Carvalho, 2009).

O protocolo foi descrito textualmente, especificado utilizando uma gramática formal e validado utilizando uma ferramenta que faz verificação utilizando técnicas formais de análise de alcançabilidade de nós em uma árvore de possibilidades. O resultado dessa validação formal indicou que o protocolo é simples porém robusto para o cenário proposto.

Uma análise mais atenta ao modelo criado permite enxergar que o modelo pode ser facilmente adaptado para outros domínios que não o aéreo, o que expande o leque de opções de trabalhos futuros gerados por esse trabalho de mestrado.

Os resultados obtidos com os testes utilizando a plataforma de prototipação Arduino mostraram que pode-se implementar o protocolo com sucesso em um *hardware* modesto, o que não acarretaria altos custos de implantação em sistemas comerciais.

A viabilização completa deste protocolo e a proposição de um protótipo de *hardware* para a sua utilização vão ao encontro do objetivo almejado no início do trabalho, que era contribuir com o progresso da autonomização destes veículos, e também para a sua proliferação, por meio do provimento de uma interface para interligação de plataformas heterogêneas.

6.1 Dificuldades Encontradas

A primeira dificuldade foi a de encontrar métodos formais para se projetar protocolos de comunicação. Foram várias as metodologias encontradas, mas poucas era confiáveis ou livres de problemas. A maioria não obedecia um processo, mas sim apenas algumas atividades específicas para algum protocolo específico.

Outra dificuldade foi elaborar o próprio projeto do protocolo SSP. Foi necessário pesquisar sobre todo o ambiente de operação que ele seria aplicado para fazer um levantamento das características desejadas e das não desejadas. A pesquisa sobre esse cenário foi a parte mais demorada do projeto, visto que existe bastante material de instituições governamentais, porém, não possuem uma padronização técnica para que pudessem ser facilmente comparados.

Uma dificuldade encontrada à época dos experimentos foi durante a implementação das aplicações que rodaram nos computadores que simulavam as entidades. Essas aplicações comunicavam-se uma com a outra, através das primitivas SSP, enviadas via Arduinos. Elas foram implementadas na linguagem Java. A biblioteca Java que faz comunicação com dispositivos seriais é a *librxtx-java*. Essa biblioteca possui um problema grave, e fazia com que a máquina virtual travasse quando os dois processos (MOSA e VANT) eram rodados no mesmo computador. O problema foi contornado quando conectou-se cada placa Arduino a um computador diferente.

O mais desafiador foi criar todo o mecanismo baseando-se em premissas já existentes, mas com requisitos distintos, visto que não foi encontrado nenhum trabalho diretamente relacionado para o ambiente de embarcados críticos complexos. Não foi encontrado na literatura nenhum trabalho que houvesse separado em duas camadas o processamento de missão e o controle da aeronave, o que conduziu a produção do trabalho publicado em (Pires et al., 2011).

6.2 Contribuições

Acredita-se que a integração entre plataformas heterogêneas proporcionada pelo protocolo SSP é um propulsor para uma padronização na comunicação entre os dispositivos envolvidos, o que propicia também a evolução da autonomização desses robôs e sua maior inserção em tarefas onde são ideais, tais como locais perigosos ou inacessíveis por pessoas e missões de longuíssima duração.

Houve também uma contribuição com a evolução deste tipo de tecnologia, visto que foi elaborado, formalmente especificado e validado um protocolo para permitir a interoperabilidade entre plataformas de diferentes fabricantes. Esta separação também facilitará a certificação dessas aeronaves para o uso do espaço aéreo.

O modo como a gramática do protocolo foi especificada também é mais uma contribuição, pois possui uma abertura para fácil adaptação, sendo possível expandí-la para outras plataformas de veículos autônomos.

6.3 Produção Científica

Artigos

- RODRIGUES, D. ; PIRES, R. M. ; ESTRELLA, J. C. ; VIEIRA, M. ; CORREA, M. ; CAMARGO JUNIOR, J. B. ; BRANCO, K. R. L. J. C. ; TRINDADE JUNIOR, O. *Application of SOA in Safety-Critical Embedded Systems*. Communications in Computer and Information Science (Print), v. 206, p. 345-354, 2011.

- PIRES, R. M. ; RODRIGUES, D. ; BRANCO, K. R. L. J. C. ; TRINDADE JUNIOR, O. *MOSA – Mission Oriented Sensor Array: A Proposal*. In: XXXVII Conferencia Latinoamericana de Informática - CLEI, 2011, Quito. Proceedings of the XXXVII Conferencia Latinoamericana de Informática - CLEI, 2011. p. 1309-1318.

- RODRIGUES, D. ; PIRES, R. M. ; ESTRELLA, J. C. ; MARCONATO, E. A. ; TRINDADE JUNIOR, O. ; BRANCO, K. R. L. J. C. . *Using SOA in Critical-Embedded Systems*. In: 2011 IEEE International Conferences on Internet of Things and Cyber, Physical and Social Computing - iThings/CPSCom, 2011, Dalian. 2011 International Conference on Internet of Things (iThings/CPSCom) and 4th International Conference on Cyber, Physical and Social Computing, 2011. p. 733-738.

Resumo

- CHAVES, A. A. ; RODRIGUES, D. ; PIRES, R. M. ; BRANCO, K. R. L. J. C. *CritServices — Desenvolvimento se Serviços para Sistemas Embarcados Críticos — Foco em Aviônicos*. Simpósio de Iniciação Científica da USP (SIICUSP), 2012.

6.4 Sugestão de trabalhos futuros

Em complemento ao trabalho realizado aqui e continuação da contribuição para a autonomização dessas aeronaves não tripuladas faz-se aqui sugestões de trabalhos futuros:

1. Aplicação de uma etapa de autenticação do processador de missão pela aeronave. Deste modo o *middleware* da aeronave também conseguiria selecionar (ou filtrar) os processadores de missão que pudessem utilizá-la como meio de transporte. Este mecanismo de autenticação seria interessante, por exemplo, em aplicação militar, onde as aeronaves podem carregar armamento e, portanto, não devem ser utilizadas sem a devida autorização;

2. Aplicação de uma camada de segurança na comunicação entre as entidades e a avaliação de sua influência no desempenho da comunicação. A segurança é um item

sempre interessante, principalmente em cenários críticos onde pode haver a perda de altos valores ou até mesmo vidas humanas;

3. Avaliação do protocolo adicionado das camadas de segurança em ambientes de execução mais robustos, como as placas Gumstix Overo e Raspberry Pi, por exemplo, visto que estes *hardwares* têm sido alvos de pesquisas de empresas interessadas em aplicações para VANTs[1] [2];

4. Utilizar o protocolo SSP como *middleware* em uma integração de um processador inteligente de missão com o módulo "Aeronave" da plataforma PIPE-SEC (Gil et al., 2009), onde as duas partes trocam mensagens para se comunicar;

5. Fazer a integração do MOSA com o *Knowledge-Based Framework* (Rodrigues et al., 2011a) para a utilização inteligente de serviços em aeronaves;

6. Por último, seria interessante validar o protocolo SSP em ambiente real de operação, executando uma missão real.

[1] `www.amazon.com/b?node=8037720011`
[2] `www.gumstix.com/gumstix-user-projects/flight-of-the-gumstix/`

Referências

ALBUS, J.; HUANG, H.; LACAZE, A.; SCHNEIER, M.; JUBERTS, M.; SCOTT, H.; BALAKIRSKY, S.; SHACKLEFORD, P. W.; HONG, T.; MICHALOSKI, J.; PROCTOR, F.; SHACKLEFORD, W.; WAVERING, A.; KRAMER, T.; LEGOWIK, S.; DAGALAKIS, N.; RIPPEY, W.; STOUFFER, K.; BOSTELMAN, R.; EVANS, J.; JACOFF, A.; NORCROSS, R.; FALCO, J.; SZABO, S.; GILSINN, J.; BUNCH, R.; TSAI, T.-M.; CHANG, T.; MEYSTEL, A.; BARBERA, A.; FITZGERALD, M. L.; GIORNO, M.; FINKELSTEIN, R.; MESSINA, E.; MURPHY, K. 4D/RCS: A Reference Model Architecture For Unmanned Vehicle Systems Version 2.0. Reference Model, NIST – Intelligent Systems Division, 2002.

ALDRIDGE JR, E. C.; STENBIT, J. P. *Unmanned Aerial Vehicles Roadmap 2002-2027*. Report, 2002.

ARDUINO SA Arduino mega 2560: Overview. Overview, 2013. Disponível em: `http://arduino.cc/en/Main/ArduinoBoardMega2560`

BEHRMANN, G.; DAVID, A.; LARSEN, K. G. A tutorial on UPPAAL. In: BERNARDO, M.; CORRADINI, F., eds. *Formal Methods for the Design of Real-Time Systems: International School on Formal Methods for the Design of Computer, Communication, and Software Systems*, v. 3185 de *Lecture Notes in Computer Science*, Bertinora, Italy: Springer Berlin Heidelberg, p. 200–236, revised Lectures, 2004.

BENGTSSON, J.; GRIFFIOEN, W. O. D.; KRISTOFFERSEN, K. J.; LARSEN, K. G.; LARSSON, F.; PETTERSSON, P.; YI, W. Automated analysis of an audio control protocol using UPPAAL. *Journal of Logic and Algebraic Programming*, v. 52–53, p. 163–181, 2002. Disponível em: `http://www.it.uu.se/research/group/darts/papers/texts/bgkllpw-jlap02.pdf`

BONAVENTURE, O. *Computer networking : Principles, protocols and practice*. 1 ed. Belgium: Université Catholique de Louvain, 278 p., 2012.

BORDBAR, B.; ANANE, R.; OKANO, K. An evaluation mechanism for QoS management in wireless systems. In: *proceedings of the 11th International Conference on Parallel and Distributed Systems, ICPADS 2005*, 2005, p. 150–154.

BRAGA, R.; BRANCO, K. R. L. J. C.; TRINDADE JR, O.; MASIERO, P. C.; NERIS, L. O.; BECKER, M. ProLICES: An approach to develop Product Lines for Safety-Critical Embedded Systems. In: *Proceedings of the XXXVII Latin-American Informatics Conference (CLEI XXXVII)*, Quito-Peru, 2011, p. 1–16.

CAMBONE, S. A.; KRIEG, K. J.; PACE, P.; WELLS II, L. *Unmanned aircraft systems roadmap 2005-2030*. Report, 2005.

CARVALHO, J. S. *Verificação Automatizada de Sistemas de Tempo Real Críticos*. Dissertação de mestrado, Universidade da Beira Interior, Portugal, 2009.

CEBUS INDUSTRY COUNCIL $\mathrm{HomePNP}^{\mathrm{TM}}$ specification version 1.0. $\mathrm{HomePlug\&Play}^{\mathrm{TM}}$*: CAL-Based Interoperability for Home Systems, Indianapolis, Ind*, p. 1–111, 1998.

CLAPPER JR, J. R.; YOUNG JR, J. J.; CARTWRIGHT, J. E.; GRIMES, J. G. *Unmanned Systems Roadmap 2007-2032*. Report, 2007.

CLAPPER JR, J. R.; YOUNG JR, J. J.; CARTWRIGHT, J. E.; GRIMES, J. G.; PAYTON, S. C.; STACKLEY, S. J.; POPPS, D. *Unmanned systems integrated roadmap fy2009–2034*. Report, Office of the Secretary of Defense, 2009.

COLOMINA, I.; BLÁZQUEZ, M.; MOLINA, P.; PARÉS, M. E.; M. WIS Towards a New Paradigm for High-Resolution Low-Cost Photogrammetry and Remote Sensing. *International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, v. XXXVII, n. B1, p. 1201–1206, 2008.

COOPER, J.; GOODRICH, M. A. Towards combining UAV and sensor operator roles in UAV-enabled visual search. In: *Proceedings of the 3rd ACM/IEEE international conference on Human robot interaction*, New York, NY, USA: ACM, 2008, p. 351–358 (*HRI '08*, v.).

CORDEIRO, G.; FILHO, V. A.; RISO, B. G. Aspectos de análise e síntese de protocolos, Universidade Federal de Santa Catarina, INE/CTC/UFSC, 2003.

DAVID, A.; YI, W. Modelling and analysis of a commercial field bus protocol. In: *Proceedings of the 12th Euromicro Conference on Real Time Systems*, IEEE Computer Society, 2000, p. 165–172.

DLNA ORGANIZATION *DLNA Networked Device Interoperability Guidelines: Architectures and Protocols*. Relatório Técnico March, 2006.

DONLEY, M. B.; SCHWARTZ, N. A. *United States Air Force Unmanned Aircraft Systems Flight Plan 2009-2047*. Report, United States Air Force, Whashington, DC, 2009.

DRURY, J. L.; RIEK, L.; RACKLIFFE, N. A decomposition of UAV-related situation awareness. In: *Proceedings of the 1st ACM SIGCHI/SIGART conference on Human-robot interaction*, New York, NY, USA: ACM, 2006, p. 88–94 (*HRI '06*, v.).

FERREIRA, N. F. G. A. *Verificação formal de sistemas modelados em estados finitos*. Dissertação de mestrado, Universidade de São Paulo. Escola Politécnica., 2005.

FERRELL, T. K.; FERRELL, U. D. *RTCA DO-178B/EUROCAE ED-12B*, cáp. 27. The Electrical Engineering Handbook Series New York, USA: CRC Press, 2001.

FONTANARI, A. A. L. *Sistema de planejamento e controle de missão de um veículo aéreo não tripulado aplicado em redes de sensores sem fio*. Tcc, Universidade Federal do Rio Grande do Sul, 2011.

GALTON, A. Temporal logic. In: ZALTA, E. N., ed. *The Stanford Encyclopedia of Philosophy*, fall 2008 ed, 2008.

GASPAR, T.; OLIVEIRA, P.; SILVESTRE, C. Uav-based marine mammals positioning and tracking system. In: *Mechatronics and Automation (ICMA), 2011 International Conference on*, 2011, p. 1050–1055.

GEORGE, J.; P. B., S.; SOUSA, J. B. Search strategies for multiple uav search and destroy missions. *J. Intell. Robotics Syst.*, v. 61, p. 355–367, 2011.

GIL, F. D. O.; CAMARGO JÚNIOR, J. A. B.; ALMEIDA JÚNIOR, J. R. D.; VISMARI, L. F.; CUGNASCA, P. S.; GIMENES, R. A. V.; FURTADO, V. H. PIPE-SEC: Platform for Testing and Validation of the Operation of Unmanned Aerial Vehicles (UAVs) in Controlled Airspace. In: *VIII SITRAER/II RIDITA, 2009*, São Paulo, Brazil, 2009, p. 508–521.

HAVELUND, K.; SKOU, A.; LARSEN, K. G.; LUND, K. Formal modelling and analysis of an audiovideo protocol: An industrial case study using uppaal. In: *Proceedings of the 18th IEEE Real-Time Systems Symposium*, San Francisco, California, USA, 1997, p. 2–13.

HESSEL, A.; PETTERSSON, P. Model-based testing of a WAP gateway: an industrial study. In: *Proceedings of FMICS and PDMC 2006, LNCS 4346*, 2006.

HODGSON, A. J.; NOAD, M.; MARSH, H.; LANYON, J.; KNIEST, E. *Using unmanned aerial vehicles for surveys of marine mammals in australia: test of concept*. Final report to the australian marine mammal centre, University of Queensland and James Cook University and University of Newcastle, Australian, 2010.

HURA, G. S.; SINGHAL, M. *Data and computer communications: networking and internetworking*. Boca Raton, FL, USA: CRC Press, Inc., 2001.

ISO LOTOS – A Formal Description Technique Based on the Temporal Ordering of Observational Behaviour. ISO/IEC 8807:1989, 1989.

ISO ESTELLE – A Formal Description Technique Based on an Extended State Transition Model. ISO/IEC 9074:1997, 1997.

ITU Specification and Description Language: ITU Z.100-Z.109. 1999.

JACQUES, D. Modeling considerations for wide area search munition effectiveness analysis. In: *Simulation Conference, 2002. Proceedings of the Winter*, 2002, p. 878 – 886 vol.1.

JORGE, L. A. D. C.; INAMASU, R. Y.; CARMO, R. B. Desenvolvimento de um VANT totalmente configurado para aplicações em Agricultura de Precisão no Brasil. *Anais XV Simpósio Brasileiro de Sensoriamento Remoto - SBSR*, , n. 1979, p. 399–406, 2011.

KARAMAN, S.; RASMUSSEN, S.; KINGSTON, D. B.; FRAZZOLI, E. Specification and Planning of UAV Missions: A Process Algebra Approach. *Science*, p. 1442–1447, 2009.

KNUTH, D. E. *Selected papers on computer languages*. N. 139 in CSLI Lecture Notes, 2nd, ilustrated ed. CSLI Publications, 594 p., 2003.

KOSKI, W. R.; ALLEN, T.; IRELAND, D.; BUCK, G.; SMITH, P. R.; MACRANDER, A. M.; HALICK, M. A.; RUSHING, C.; SLIWA, D. J.; MCDONALD, T. L. Evaluation of an Unmanned Airborne System for Monitoring Marine Mammals. *Aquatic Mammals*, v. 35, n. 3, p. 347–357, 2009.

LINDAHL, M.; PETTERSSON, P.; YI, W. Formal Design and Analysis of a Gear-Box Controller. In: *Proc. of the 4*th *Workshop on Tools and Algorithms for the Construction and Analysis of Systems*, Springer–Verlag, 1998, p. 281–297 (*Lecture Notes in Computer Science*, v.).

LINDAHL, M.; PETTERSSON, P.; YI, W. Formal Design and Analysis of a Gearbox Controller. *Springer International Journal of Software Tools for Technology Transfer (STTT)*, v. 3, n. 3, p. 353–368, 2001.

LÖNN, H.; PETTERSSON, P. Formal Verification of a TDMA Protocol Startup Mechanism. In: *Proc. of the Pacific Rim Int. Symp. on Fault-Tolerant Systems*, 1997, p. 235–242.

MAZA, I.; KONDAK, K.; BERNARD, M.; OLLERO, A. Multi-uav cooperation and control for load transportation and deployment. *Journal of Intelligent and Robotic Systems*, v. 57,

n. 1-4, p. 417–449, 2010.
Disponível em: http://dx.doi.org/10.1007/s10846-009-9352-8

MERINO, L.; CABALLERO, F.; DIOS, J.; FERRUZ, J.; OLLERO, A. A cooperative perception system for multiple uavs: Application to automatic detection of forest fires. *Journal of Field Robotics*, v. 23, n. 3-4, p. 165–184, 2006.

MICA, J.; COSTELLO, J. F. *Unmanned aircraft systems: Federal actions needed to ensure safety and expand their potential uses within the national airspace system.* Report, 2008.

MICROSOFT *Understanding Universal Plug and Play: White Paper*. Relatório Técnico, Redmond, 45p, 2000.

MOECKE, M.; FARINES, J. M. Árvore de alcançabilidade para redes predicado-transição: um método de redução que preserva as propriedades de análise. In: *Anais do VI Simpósio Brasileiro de Engenharia de Software - SBES*, Gramado, RS: UFRGS, 1992, p. 157–170.

MOURA, J.; SAUVÉ, J.; GIOZZA, W.; ARAÚJO, J. *Redes Locais de Computadores: Protocolos de Alto Nível e Avaliação de Desempenho*. 1 ed. São Paulo, Brasil: McGraw-Hill, 446 p., 1986.

MULLENS, K. D.; PACIS, E. B.; STANCLIFF, S. B.; BURMEISTER, A. B.; SAIC, T. A. D.; BRUCH, M. H.; EVERETT, H. R. An Automated UAV Mission System. *AUVSI Unmanned Systems in International Security 2003 USIS 03*, 2003.
Disponível em: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.11.824&rep=rep1&type=pdf

NASA Nasa airborne science program. Webpage, 2011.

O'DRISCOLL, G. *The essential guide to home networking technologies*, cáp. 15 - HomePnP. Essential Guide Series Prentice Hall, p. 193–199, 2000.

PIRES, R. M.; RODRIGUES, D.; BRANCO, K. R. L. J. C.; TRINDADE JR, O. MOSA – Mission Oriented Sensors Array: A Proposal. In: *Proceedings of the XXXVII Conferencia Latinoamericana de Informática – CLEI*, Quito, Ecuador, 2011, p. 1309–1318.

PITRE, R.; LI, X.; DELBALZO, D. A new performance metric for search and track missions 2: Design and application to uav search. In: *FUSION '09. 12th International Conference on Information Fusion, 2009.*, 2009, p. 1108–1114.

RAVN, A.; SRBA, J.; VIGHIO, S. Modelling and verication of web services business activity protocol. In: *Proceedings of the 17th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, 2010a.

RAVN, A.; VIGHIO, S.; SRBA, J. A formal analysis of the web services atomic transaction protocol with uppaal. In: *Proceedings of the 4th International Symposium On Leveraging Applications of Formal Methods, Verification and Validation (ISOLA'10)*, Springer-Verlag, 2010b.

RODRIGUES, D.; ESTRELLA, J. C.; VIEIRA, M.; CAMARGO JÚNIOR, J. B.; BRANCO, K. R. L. J. C.; TRINDADE JR, O. Service-Oriented Architectures for Complex Safety-Critical Embedded Systems: A Case Study on UAVs. In: *Proceedings of the I Brazilian Conference on Critical Embedded Systems – CBSEC*, são Carlos-SP, Brazil, 2011a, p. 130–130.

RODRIGUES, D.; PIRES, R. M.; ESTRELLA, J. C.; MARCONATO, E. A.; TRINDADE JR, O.; BRANCO, K. R. L. J. C. Using SOA in Critical-Embedded Systems. In: *2011 International Conference on Internet of Things and 4th International Conference on Cyber, Physical and Social Computing*, IEEE, 2011b, p. 733–738.

SHARIFF, A. R. M.; SHASHIKANT, V.; NOURQOLIPOUR, R. Integration of uav with stress detection lens over oil palm plantation. In: *The 33rd Asian Conference on Remote Sensing - ACRS*, 2012.

SUN *Jini tm architecture specification*. Relatório Técnico, Palo Alto, 26p, 2011.

TARNAY, K. *Protocol specification and testing*. illustrated ed. University of Michigan: Plenum Press, 367 p., 1991.

THRAMBOULIDIS, K. C.; DOUKAS, G.; KOUMOUTSOS, G. A SOA-based Embedded Systems Development Environment For Industrial Automation. *EURASIP J. Embedded Syst.*, v. 2008, p. 3:1–3:15, 2008.

TRINDADE JR, O.; BARBOSA, L. C. P.; NERIS, L. D. O.; JORGE, L. A. C. A mission planner and navigation system for the arara project. *23rd ICAS - Congress of International Council of the Aeronautical Sciences*, 2002.

TRINDADE JR, O.; BRAGA, R. T. V.; NERIS, L. D. O.; BRANCO, K. R. L. J. C. Uma Metodologia para Desenvolvimento de Sistemas Embarcados Críticos com Vistas à Certificação. *Anais do IX Simpósio de Automação Inteligente - IX SBAI*, p. 1–6, 2009.

TRINDADE JR, O.; BRANCO, K. R. L. J. C.; NERIS, L. D. O.; CHAVIER, L. F. C. Robôs Aéreos. In: ROMERO, R. A. F.; OSÓRIO, F. S.; WOLF, D. F.; SILVA JR, E. P. E., eds. *Robótica Móvel*, cáp. 16, LTC, p. 461–478, 2012.

TRINDADE JR, O.; OLIVEIRA NERIS, L.; BARBOSA, L. C. P.; BRANCO, K. R. L. J. C. A layered approach to design autopilots. In: *IEEE International Conference on Industrial Technology - ICIT*, 2010, p. 1415 –1420.

TSO, K. S.; THARP, G. K.; ZHANG, W.; TAI, A. T. A multi-agent operator interface for unmanned aerial vehicles. In: *Digital Avionics Systems Conference, 1999. Proceedings. 18th*, 1999, p. 6.A.4–1 –6.A.4–8 vol.2.

TURNER, K. J., ed. *Using formal description techniques: An introduction to estelle, lotos and sdl*. Wiley series in communication and distributed systems. Wiley, 460 p., 1993.

UPPAAL TEAM Uppaal: About. `http://www.it.uu.se/research/group/darts/uppaal/about.shtml`, 2013.

U.S. ARMY *Unmanned Aircraft Systems Roadmap 2010-2035: Eyes of the Army*. Relatório Técnico, 2010.

VALAVANIS, K. P. *Advances In Unmanned Aerial Vehicles: State Of The Art And The Road to Autonomy*, v. 33 de *International series on intelligent systems, control, and automation*. Springer, 2007.

WANG, J.; PATEL, V.; WOOLSEY, C.; HOVAKIMYAN, N.; SCHMALE, D. L1 adaptive control of a uav for aerobiological sampling. In: *American Control Conference, 2007. ACC '07*, 2007, p. 4660–4665.

WILLIGEN, W. H.; SCHUT, M. C.; EIBEN, A. E.; KESTER, L. J. H. M. Online adaptation of path formation in uav search-and-identify missions. In: *Proceedings of the 10th international conference on Adaptive and natural computing algorithms - Volume Part II*, Berlin, Heidelberg: Springer-Verlag, 2011, p. 186–195 (*ICANNGA'11*, v.).

ZHANG, C.; KOVACS, J. The application of small unmanned aerial systems for precision agriculture: a review. *Precision Agriculture*, v. 13, n. 6, p. 693–712, 2012.

APÊNDICE

A

Gramática do Protocolo SSP, em BNF

⟨*REQUEST*⟩ ::= ‘REQ’ ⟨*FeatureName*⟩

⟨*FeatureName*⟩ ::= ‘Env’ | ‘Lum’ | ‘Wea’ | ‘Alb’ | ‘Spb’ | ‘End’ | ‘Sec’

⟨*RETURN*⟩ ::= ‘RET’ ⟨*Feature*⟩

⟨*Feature*⟩ ::= ⟨*Environment*⟩ | ⟨*Luminosity*⟩ | ⟨*Weather*⟩ | ⟨*AltitudeBoundaries*⟩ | ⟨*SpeedBoundaries*⟩ | ⟨*Endurance*⟩ | ⟨*Secrecy*⟩

⟨*Environment*⟩ ::= ‘air’ | ‘gnd’ | ‘sfc’ | ‘uns’

⟨*Luminosity*⟩ ::= ‘dlg’ | ‘ngt’ | ‘bot’

⟨*Weather*⟩ ::= ⟨*Sunny*⟩

⟨*Sunny*⟩ ::= ‘sun’ | ‘sun’ ⟨*Cloudy*⟩ | ⟨*Cloudy*⟩

⟨*Cloudy*⟩ ::= ‘clo’ | ‘clo’ ⟨*Stormy*⟩ | ⟨*Stormy*⟩

⟨*Stormy*⟩ ::= ‘sto’ | ‘sto’ ⟨*Rainy*⟩ | ⟨*Rainy*⟩

⟨*Rainy*⟩ ::= ‘rai’ | ‘rai’ ⟨*Windy*⟩ | ⟨*Windy*⟩

⟨*Windy*⟩ ::= ‘win’ | ‘win’ ⟨*Snowy*⟩ | ⟨*Snowy*⟩

⟨*Snowy*⟩ ::= ‘sno’ | ‘sno’ ⟨*Foggy*⟩ | ⟨*Foggy*⟩

⟨*Foggy*⟩ ::= ‘fog’ | ‘fog’ ⟨*Hot*⟩ | ⟨*Hot*⟩

⟨*Hot*⟩ ::= 'hot' | 'hot' ⟨*Cold*⟩ | ⟨*Cold*⟩

⟨*Cold*⟩ ::= 'col' | 'col' ⟨*Wet*⟩ | ⟨*Wet*⟩

⟨*Wet*⟩ ::= 'wet' | 'wet' ⟨*Dry*⟩ | ⟨*Dry*⟩

⟨*Dry*⟩ ::= 'dry'

⟨*AltitudeBoundaries*⟩ ::= ⟨*MinAltitude*⟩ ⟨*MaxAltitude*⟩

⟨*MinAltitude*⟩ ::= ⟨*Altitude*⟩ *% less than* <MaxAltitude> *altitude;*

⟨*MaxAltitude*⟩ ::= ⟨*Altitude*⟩ *% greater than* <MinAltitude> *altitude;*

⟨*Altitude*⟩ ::= ⟨*integer*⟩ *% Positive integer, greater than zero;*

⟨*SpeedBoundaries*⟩ ::= ⟨*MinSpeed*⟩ ⟨*MaxSpeed*⟩

⟨*MinSpeed*⟩ ::= ⟨*Speed*⟩ *% less than* <MaxSpeed> *speed;*

⟨*MaxSpeed*⟩ ::= ⟨*Speed*⟩ *% greater than* <MinSpeed> *speed;*

⟨*Speed*⟩ ::= ⟨*integer*⟩ *% Positive integer, greater than zero;*

⟨*Endurance*⟩ ::= 'min' | 'hrs' | 'day'

⟨*Secrecy*⟩ ::= 'ost' | 'sec' | 'top'

⟨*SEND*⟩ ::= 'SND' ⟨*Length*⟩ **data**

⟨*Length*⟩ ::= ⟨*integer*⟩ *% Positive integer, greater than zero;*

⟨*ACK*⟩ ::= 'ACK' ⟨*InstructionName*⟩

⟨*InstructionName*⟩ ::= 'SND' | 'GTO' | 'NTF' | 'LND' | 'CLS'

⟨*GOTO*⟩ ::= 'GTO' ⟨*Waypoint*⟩

⟨*Waypoint*⟩ ::= ⟨*ID*⟩ ⟨*Priority*⟩ ⟨*Latitude*⟩ ⟨*Longitude*⟩ ⟨*AltitudeBoundaries*⟩ ⟨*Speed*⟩

⟨*ID*⟩ ::= ⟨*integer*⟩ *% Sequential positive integer;*

⟨*Priority*⟩ ::= ⟨*Normal*⟩ | ⟨*Medium*⟩ | ⟨*High*⟩ | ⟨*Higher*⟩

⟨*Normal*⟩ ::= 'NOR'

⟨*Medium*⟩ ::= 'MED'

⟨*High*⟩ ::= 'HIG'

⟨*Higher*⟩ ::= 'HGR'

⟨*Latitude*⟩ ::= ⟨*decimal*⟩ *% Decimal number in range [-90,90], inclusive.*
% The latitude is preceded by a minus sign if it is south of
% the equator (a positive number implies north);

⟨*Longitude*⟩ ::= ⟨*decimal*⟩ *% Decimal number in range [-180,180], inclusive.*
% The longitude is preceded by a minus sign if it is west of
% the prime meridian (a positive number implies east);

⟨*NOTIFY*⟩ ::= 'NTF' ⟨*WPid*⟩

⟨*WPid*⟩ ::= ⟨*ID*⟩

⟨*LAND*⟩ ::= 'LAN' ⟨*Waypoint*⟩ ⟨*Heading*⟩

⟨*Heading*⟩ ::= ⟨*integer*⟩ *% Positive integer in range [1,36], inclusive, which is one*
% tenth of the magnetic azimuth of the runway's heading,
% in degrees;

⟨*CLOSE*⟩ ::= 'CLS'

⟨*ABORT*⟩ ::= 'ABT' *% Only allowed to be sent by the aircraft;*

⟨*integer*⟩ ::= **int16** *% 2-byte signed integer number;*

⟨*decimal*⟩ ::= **float32** *% 4-byte signed float number;*
% 23 bits of mantissa, 8 bits exponent and 1 bit sign

APÊNDICE

B

Gramática de uma missão de exemplo, em BNF

Abaixo segue a descrição da gramática de missão utilizada neste trabalho, adaptada de (Fontanari, 2011).

⟨*Mission*⟩ ::= ⟨*Name*⟩ ⟨*Description*⟩ ⟨*Priority*⟩ ⟨*SensorType*⟩ ⟨*MissionType*⟩ ⟨*Endurance*⟩ ⟨*Secrecy*⟩ ⟨*Climate*⟩ ⟨*WPlist*⟩

⟨*Name*⟩ ::= ⟨*string*⟩ *% Maximum string length here is 30;*

⟨*Description*⟩ ::= ⟨*string*⟩ *% Maximum string length here is 256;*

⟨*Priority*⟩ ::= ⟨*Normal*⟩ | ⟨*Medium*⟩ | ⟨*High*⟩ | ⟨*Higher*⟩

⟨*SensorType*⟩ ::= ⟨*Active*⟩ | ⟨*Passive*⟩ | ⟨*Both*⟩

⟨*MissionType*⟩ ::= ⟨*SeqPictures*⟩ | ⟨*Pictures*⟩ | ⟨*Videos*⟩ | ⟨*Sensing*⟩

⟨*Endurance*⟩ ::= ⟨*Minutes*⟩ | ⟨*Hours*⟩ | ⟨*Days*⟩

⟨*Secrecy*⟩ ::= ⟨*Ostensive*⟩ | ⟨*Secret*⟩ | ⟨*TopSecret*⟩

⟨*Climate*⟩ ::= ⟨*Weather*⟩ ⟨*Luminosity*⟩ ⟨*Environment*⟩

⟨*WPlist*⟩ ::= ⟨*Takeoff*⟩ ⟨*Waypoint*⟩* ⟨*Land*⟩

⟨*Normal*⟩ ::= ‘n’

⟨*Medium*⟩ ::= ‘m’

⟨*High*⟩ ::= ‘h’

⟨*Higher*⟩ ::= ‘H’

⟨*Active*⟩ ::= ‘a’

⟨*Passive*⟩ ::= ‘p’

⟨*Both*⟩ ::= ‘b’

⟨*SeqPictures*⟩ ::= ‘P’

⟨*Pictures*⟩ ::= ‘i’

⟨*Videos*⟩ ::= ‘v’

⟨*Sensing*⟩ ::= ‘s’

⟨*Minutes*⟩ ::= ‘e’

⟨*Hours*⟩ ::= ‘r’

⟨*Days*⟩ ::= ‘y’

⟨*Ostensive*⟩ ::= ‘o’

⟨*Secret*⟩ ::= ‘t’

⟨*TopSecret*⟩ ::= ‘T’

⟨*Weather*⟩ ::= ⟨*Sunny*⟩

⟨*Sunny*⟩ ::= ‘sun’ | ‘sun’ ⟨*Cloudy*⟩ | ⟨*Cloudy*⟩

⟨*Cloudy*⟩ ::= ‘clo’ | ‘clo’ ⟨*Stormy*⟩ | ⟨*Stormy*⟩

⟨*Stormy*⟩ ::= ‘sto’ | ‘sto’ ⟨*Rainy*⟩ | ⟨*Rainy*⟩

⟨*Rainy*⟩ ::= ‘rai’ | ‘rai’ ⟨*Windy*⟩ | ⟨*Windy*⟩

⟨*Windy*⟩ ::= ‘win’ | ‘win’ ⟨*Snowy*⟩ | ⟨*Snowy*⟩

⟨*Snowy*⟩ ::= ‘sno’ | ‘sno’ ⟨*Foggy*⟩ | ⟨*Foggy*⟩

⟨*Foggy*⟩ ::= ‘fog’ | ‘fog’ ⟨*Hot*⟩ | ⟨*Hot*⟩

⟨*Hot*⟩ ::= ‘hot’ | ‘hot’ ⟨*Cold*⟩ | ⟨*Cold*⟩

⟨*Cold*⟩ ::= '`col`' | '`col`' ⟨*Wet*⟩ | ⟨*Wet*⟩

⟨*Wet*⟩ ::= '`wet`' | '`wet`' ⟨*Dry*⟩ | ⟨*Dry*⟩

⟨*Dry*⟩ ::= '`dry`'

⟨*Luminosity*⟩ ::= ⟨*Daylight*⟩ | ⟨*Night*⟩ | ⟨*Both*⟩

⟨*Environment*⟩ ::= ⟨*Air*⟩ | ⟨*Ground*⟩ | ⟨*Surface*⟩ | ⟨*Undersea*⟩

⟨*Daylight*⟩ ::= '`d`'

⟨*Night*⟩ ::= '`g`'

⟨*Air*⟩ ::= '`A`'

⟨*Ground*⟩ ::= '`G`'

⟨*Surface*⟩ ::= '`S`'

⟨*Undersea*⟩ ::= '`U`'

⟨*Waypoint*⟩ ::= ⟨*ID*⟩ ⟨*Description*⟩ ⟨*Priority*⟩ ⟨*Latitude*⟩ ⟨*Longitude*⟩ ⟨*MinAltitude*⟩ ⟨*MaxAltitude*⟩ ⟨*Speed*⟩ ⟨*Task*⟩

⟨*Takeoff*⟩ ::= ⟨*Waypoint*⟩

⟨*Land*⟩ ::= ⟨*Waypoint*⟩ ⟨*Heading*⟩

⟨*Heading*⟩ ::= ⟨*Angle*⟩

⟨*Angle*⟩ ::= ⟨*integer*⟩ *% Positive integer in range [1,36], inclusive, which is one % tenth of the magnetic azimuth of the runway's heading, % in degrees;*

⟨*ID*⟩ ::= ⟨*integer*⟩ *% Sequential positive integer;*

⟨*Latitude*⟩ ::= ⟨*decimal*⟩ *% Decimal number in range [-90,90], inclusive. % The latitude is preceded by a minus sign if it is south of % the equator (a positive number implies north);*

⟨*Longitude*⟩ ::= ⟨*decimal*⟩ *% Decimal number in range [-180,180], inclusive. % The longitude is preceded by a minus sign if it is west of % the prime meridian (a positive number implies east);*

⟨*MinAltitude*⟩ ::= ⟨*Altitude*⟩ *% less than* `<MaxAltitude>` *altitude;*

⟨*MaxAltitude*⟩ ::= ⟨*Altitude*⟩ *% greater than* `<MinAltitude>` *altitude;*

⟨*Speed*⟩ ::= ⟨*integer*⟩ *% Positive integer, greater than zero;*

⟨*Task*⟩ ::= ⟨*Shoot*⟩ | ⟨*StartRecording*⟩ | ⟨*StopRecording*⟩ | ⟨*StartSensing*⟩ | ⟨*StopSensing*⟩ | ⟨*empty*⟩

⟨*Shoot*⟩ ::= '1'

⟨*StartRecording*⟩ ::= '2'

⟨*StopRecording*⟩ ::= '3'

⟨*StartSensing*⟩ ::= '4'

⟨*StopSensing*⟩ ::= '5'

⟨*empty*⟩ ::= ∅

⟨*string*⟩ ::= ⟨*char*⟩ ⟨*char*⟩*

⟨*char*⟩ ::= **uint16** *% 2-byte unsigned integer number;*

⟨*integer*⟩ ::= **int16** *% 2-byte signed integer number;*

⟨*decimal*⟩ ::= **float32** *% 4-byte signed float number;*
% 23 bits of mantissa, 8 bits exponent and 1 bit sign

APÊNDICE

C

Missão utilizada no Experimento 2

A Figura C.1 mostra a região de simulação do voo.

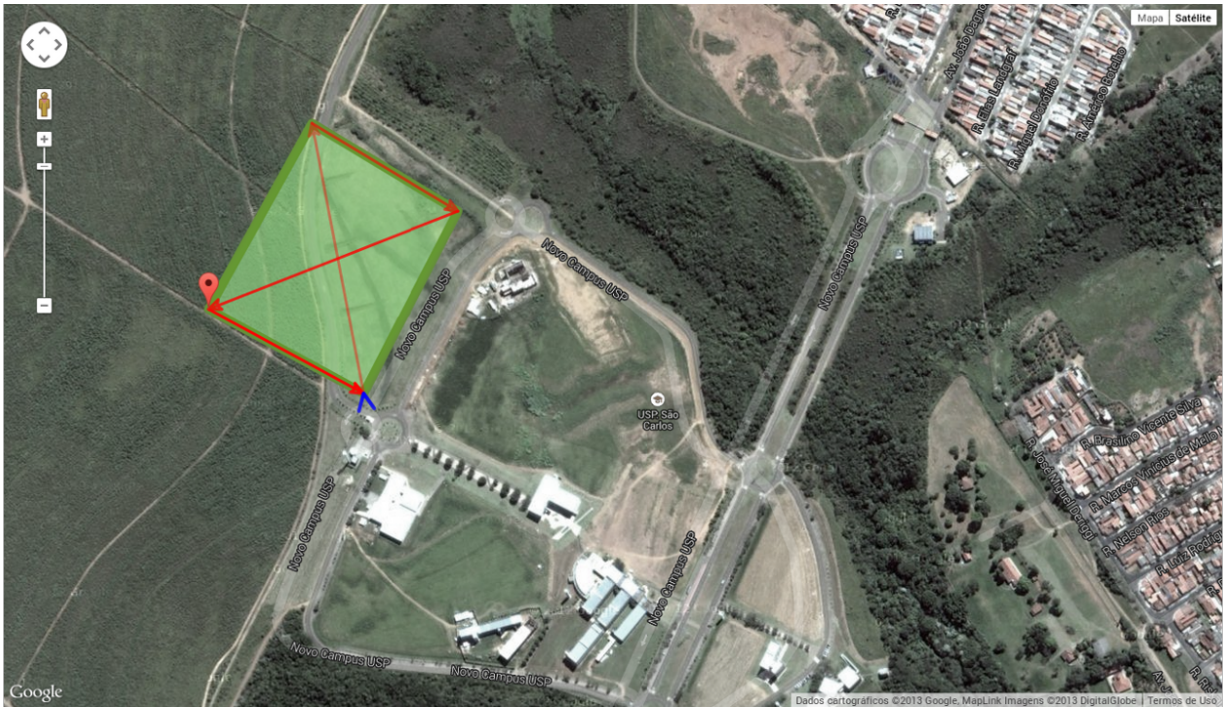


Figura C.1: Região onde foi simulado o voo da aeronave, durante os experimentos

```
<?xml version="1.0" encoding="UTF-8"?>

<Mission
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="mission.xsd">
    <Name>Missão</Name>
    <Description>Missão criada para a simulação</Description>
    <Priority>n</Priority>
    <SensorType>p</SensorType>
    <MissionType>i</MissionType>
    <Endurance>e</Endurance>
    <Secrecy>o</Secrecy>
    <Climate>
        <Weather>sun</Weather>
        <Weather>clo</Weather>
        <Weather>hot</Weather>
        <Weather>col</Weather>
        <Weather>wet</Weather>
        <Weather>dry</Weather>
        <Luminosity>d</Luminosity>
        <Environment>A</Environment>
    </Climate>
    <WPList>
        <Takeoff>
            <Waypoint>
                <Number>1</Number>
                <Description>Decolagem</Description>
                <Priority>n</Priority>
                <Latitude>-21.99818</Latitude>
                <Longitude>-47.9343</Longitude>
                <Altitude>
                    <MinAltitude>10</MinAltitude>
                    <MaxAltitude>50</MaxAltitude>
                </Altitude>
                <Speed>80</Speed>
                <Task></Task>
            </Waypoint>
        </Takeoff>
        <Waypoint>
            <Number>2</Number>
            <Description>Primeira ponta do retângulo</Description>
```

```
        <Priority>n</Priority>
        <Latitude>-21.99795</Latitude>
        <Longitude>-47.93280</Longitude>
        <Altitude>
            <MinAltitude>90</MinAltitude>
            <MaxAltitude>100</MaxAltitude>
        </Altitude>
        <Speed>70</Speed>
        <Task>2</Task>
    </Waypoint>
    <Waypoint>
        <Number>3</Number>
        <Description>Segunda ponta do retângulo</Description>
        <Priority>n</Priority>
        <Latitude>-21.99700</Latitude>
        <Longitude>-47.93450</Longitude>
        <Altitude>
            <MinAltitude>90</MinAltitude>
            <MaxAltitude>100</MaxAltitude>
        </Altitude>
        <Speed>70</Speed>
        <Task>2</Task>
    </Waypoint>
    <Waypoint>
        <Number>4</Number>
        <Description>Terceira ponta do retângulo</Description>
        <Priority>n</Priority>
        <Latitude>-21.99700</Latitude>
        <Longitude>-47.93450</Longitude>
        <Altitude>
            <MinAltitude>90</MinAltitude>
            <MaxAltitude>100</MaxAltitude>
        </Altitude>
        <Speed>70</Speed>
        <Task>2</Task>
    </Waypoint>
    <Waypoint>
        <Number>5</Number>
        <Description>Quarta ponta do retângulo</Description>
        <Priority>n</Priority>
        <Latitude>-21.99900</Latitude>
```

```
            <Longitude>-47.93570</Longitude>
            <Altitude>
                <MinAltitude>90</MinAltitude>
                <MaxAltitude>100</MaxAltitude>
            </Altitude>
            <Speed>70</Speed>
            <Task>2</Task>
        </Waypoint>
        <Waypoint>
            <Number>6</Number>
            <Description>Primeira ponta do retângulo</Description>
            <Priority>n</Priority>
            <Latitude>-21.99795</Latitude>
            <Longitude>-47.93280</Longitude>
            <Altitude>
                <MinAltitude>90</MinAltitude>
                <MaxAltitude>100</MaxAltitude>
            </Altitude>
            <Speed>70</Speed>
            <Task>3</Task>
        </Waypoint>
        <Land>
            <Waypoint>
                <Number>7</Number>
                <Description>Pouso</Description>
                <Priority>n</Priority>
                <Latitude>-21.998180</Latitude>
                <Longitude>-47.934300</Longitude>
                <Altitude>
                    <MinAltitude>0</MinAltitude>
                    <MaxAltitude>5</MaxAltitude>
                </Altitude>
                <Speed>80</Speed>
                <Task></Task>
            </Waypoint>
            <Heading>180</Heading>
        </Land>
    </WPList>
</Mission>
```