FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Metodologias e Ferramentas para o Teste e Validação de Sistemas

Filipe Daniel de Melo Ferreira

Mestrado Integrado em Engenharia Informática e Computação

Orientador: Gil Gonçalves (Eng.)

28 de Junho de 2010

Metodologias e Ferramentas para o Teste e Validação de Sistemas

Filipe Daniel de Melo Ferreira

Mestrado Integrado em Engenharia Informática e Computação

Aprovado em provas públicas pelo Júri:

Presidente: José Manuel Magalhães Cruz (Professor Auxiliar)

Vogal Externo: Gustavo Ribeiro da Costa Alves (Professor Adjunto)

Orientador: Gil Manuel Magalhães de Andrade Gonçalves (Professor Convidado)

_____

28 de Junho de 2010

Resumo

Actualmente, as empresas investem cada vez mais na segurança de sistemas críticos pois, cada vez mais riscos estes sistemas correm e cada vez mais responsabilidade lhes é associada. Sistemas como este, para o qual o projecto foi desenvolvido, são um exemplo de algo que tem de ser o mais fiável e seguro possível. No entanto é necessário que este sistema seja submetido a um rigoroso processo de testes para que não existam falhas e caso aconteçam sejam minimizadas.

Esta dissertação foi realizada no LSTS (Laboratório de Sistemas e Tecnologia Subaquática) para o projecto PITVANT , que está a ser desenvolvido em conjunto com a Força Aérea Portuguesa, e tem como missão dotar esta unidade militar de veículos autónomos não tripulados.

Estudos apontam que à medida que um sistema é testado vão sendo descobertos vários erros e existe um limite onde se torna quase impossível de testar mais esse sistema. Isso é o que este sistema trás de novo, uma ferramenta que possa ser usada para o teste e validação de sistema e para que fosse possível tornar estes testes o mais autónomos e o mais abrangentes possíveis.

No entanto existem três tipos de falhas que qualquer equipa de desenvolvimento não pode contornar. A mais grave é a falhas de requisitos, isto é a utilização do sistema em condições para o qual nunca antes foram pensados. A segunda é a super valorização dos sistemas por parte dos operadores, por exemplo a extrema confiança que os operadores apresentam quando operam um sistema destes pode fazer com que estes baixem a guarda e possam acontecer descuidos graves. E finalmente a incapacidade para agir por parte dos operadores devido à abstracção cada vez maior entre o sistema e o ser humano destes sistemas.

Este é o principal objectivo desta dissertação, apresentar uma ferramenta que possibilite colmatar estas três falhas. Este sistema tem a missão de testar e validar o sistema e preparar os operadores para situações de risco.

Os objectivos foram amplamente cumpridos pois no fecho desta dissertação esta ferramenta já tem a capacidade de testar e formar os operadores envolvidos no decurso de uma missão, no entanto existem vários objectivos que ainda não foram alcançados, como por exemplo a generalização de testes para todas as falhas que se pretendem realizar e a avaliação dos testes, que no caso de testes autónomos é necessário e de vital importância realizar.

Abstract

Nowadays critical systems are one of the most important components in IT business because of the responsibility that are putted into this hardware and software components. Systems like the one develop in the Underwater Systems and Technology Laboratory to the project PITVANT is a good example to this kind of systems that can't fail. The project aims to develop unmanned planes and systems to the Portuguese Air Force. So it is obvious the risk of causing major damages in a mission is very high. That is a risk that can't be supported by this entity and for that it is essential a good and reliable system. That is the main objective of this thesis. Ensure a good quality of the system, but also of the Flight Manager and Operator responsible for controlling the system.

Studies prove that if a system is submitted to one exhausting phase of tests the possibility of finding and fixing errors is very high and it could find almost every error in the system. But it's impossible to find every error in the system and also is impossible to have testers all day running tests in the program. That's why running simulations is one of the methods that today is being used more and more by the companies. This happens because if a system runs on the scenario that is supposed to work the testers can run several tasks to see if the system can handle it and test if everything is going as they planned.

But there are three types of errors that can't be fixed by this type of tests. The first one is that a system simulation can't be made in situations that weren't planed by the development team. This is a risk because sometimes the specification is not properly made and the system is putted in different scenarios that the ones thought to the system. The second is that the People operating critical system sometimes turn to trust too much in the system and when a problem occurs they are like sleeping and can't react fast enough to fix the problem. The third and more important problem is that today a system operator is so far away from the real system that when a real problem occurs the operator doesn't know what to do to save the system, or sometimes lives.

This is the main goal of this system, the implementation of a platform capable of running tests in simulation and at the same time being capable of making tests that humans can't predict, like injecting faults. The other main goal is to run this platform in a simulator where the human operators in a mission can learn how to respond to several types of faults, and became aware and prepared when a malfunction occurs.

Agradecimentos

Venho por este meio manifestar o meu agradecimento e apreço por todas as pessoas que contribuíram directa ou indirectamente para bom desfecho desta Dissertação.

Gostaria de focar o contributo do Professor Gil Gonçalves pela forma que contribuiu para este documento e projecto, pela forma que sempre se mostrou disponível e pela forma que sempre conduziu este projecto.

Gostaria ainda de agradecer a todas as pessoas que constituem a equipa AsasF, do Laboratório de Sistemas e tecnologia Subaquática (LSTS), pelo enorme apoio, compreensão e ajuda que sempre que necessário ofereceram. Para além desta equipa gostaria de agradecer especialmente ao Eduardo Marquês pela grande ajuda prestada na integração do sistema na plataforma Dune, sem ele este projecto seria certamente mais complicado.

Gostaria ainda de agradecer a todos os que indirectamente contribuíram para o final deste projecto, de salientar o contributo de família e amigos.

O Autor Filipe Ferreira

Índice

Lista de Figuras

Abreviaturas e Símbolos

UAV	Veículos aéreos não tripulados
PITVANT	Projecto de Investigação e Tecnologia em Veículos Aéreos Não -Tripulados
NTSB	Autoridade americana para a segurança nos transportes
Groundstation	Estação de operação terrestre
AIAA	American Institute of Aeronautics Astronautics
ANSI	American National Standards Institute
NASA	National Aeronautics and Space Administration
SGOS	Shuttle ground operations simulator
IMC	Inter-Module Communication protocol
UAS	Unmanned Aircraft System
ESA	European Space Agency

1 Introdução

Esta dissertação retrata o processo de descoberta de novas metodologias e ferramentas de suporte, para o teste e validação de componentes a integrar em sistemas de veículos aéreos autónomos não tripulados.

Este capítulo serve para introduzir todos os pontos importantes para a correcta percepção do projecto bem como alguns conceitos que serão abordados ao longo de toda a dissertação. Para isso estará dividido em vários tópicos para uma mais fácil leitura e assimilação de conceitos.

Assim sendo serão apresentados e explicados o projecto PITVANT, o que são UAV's, as motivações principais para este projecto, os objectivos do mesmo, o sistema actual e o porque da necessidade de um projecto como este.

1.1 Contexto/Enquadramento

A utilização, em teatro de operações, de veículos aéreos não tripulados equipados com tecnologia já bastante mais evoluída, teve lugar em 1982 durante o conflito Paz para a Galileia, no qual foi levada a cabo, com grande sucesso, pela Força Aérea Israelita, a destruição das defesas aéreas Sírias no Vale de Bekaa. Como consequência, muitos países, e em particular os Estados Unidos da América, reconheceram o interesse da utilização destes sistemas em operações militares, tendo decidido, em conformidade, implementar uma política dirigida no sentido do seu desenvolvimento.

Introdução

Entre 1991 e 1993, durante o conflito Tempestade no Deserto, as forças aliadas utilizaram, com elevado sucesso, vários tipos de veículos aéreos não tripulados nalgumas missões operacionais. Em meados dos anos 90, durante as operações de segurança nos conflitos da Bósnia e do Kosovo, os veículos aéreos não - tripulados vieram mostrar, mais uma vez, as suas potencialidades como sistema de armas. Mais recentemente, desde o início da guerra global contra o terrorismo, logo após os acontecimentos do 11 de Setembro de 2001, assiste-se a uma cada vez maior utilização de UAV's num largo espectro de missões militares. De facto, não tem cessado de aumentar, desde então, a contribuição destes sistemas no contexto de operações militares, quanto ao número de saídas, horas de voo acumuladas e tipos de missões desempenhadas.

Como exemplo ilustrativo desta realidade, pode afirmar-se que, desde 11 de Setembro de 2001 até Setembro de 2004, cerca de vinte tipos diferentes de UAV's – que vão desde os UAV's com menos de 1 kg de peso à descolagem e um custo de algumas centenas de euros, até aos UAV's de grandes dimensões com mais de 10 toneladas de peso à descolagem e um custo que pode ultrapassar a dezena de milhão de euros – pertencentes às forças da coligação, realizaram mais de 100.000 horas de voo no apoio às operações militares Enduring Freedom e Iraqi Freedom.

Este projecto foi desenvolvido e direccionado para o projecto PITVANT que tem como função dotar a força aérea portuguesa de aviões não tripulados, foi desenvolvido no LSTS (Laboratório de Sistemas e Tecnologia Subaquática) e por isso contou com a participação de elementos de várias equipas.

Figura 1: RQ-4 Global Hawk, um avião de altitude elevada e autonomia de 36 de voo.

1.2 Projecto PITVANT

O “Projecto de Investigação e Tecnologia em Veículos Aéreos Não -Tripulados”, doravante designado pela sigla PITVANT, é a continuação do projecto desenvolvido na Academia da força aérea desde 1996, no âmbito dos veículos aéreos autónomos não - tripulados e, fundamentalmente, o que se espera vir a fazer no futuro, no sentido de se ir dotando a Força Aérea das capacidades indispensáveis à aquisição e exploração daqueles veículos, de importância operacional cada vez maior.

O Programa de Investigação e Tecnologia em Veículos Aéreos Autónomos Não – Tripulados está dividido em várias fases, tendo como objectivos essenciais os seguintes:

Fase 1) Desenvolver protótipos de sistemas de veículos aéreos autónomos não - tripulados, essencialmente de pequena e média dimensão, integrando as mais modernas tecnologias, a usar em diversas missões e actividades, não só de natureza estritamente militar, mas também de natureza civil, incluindo naturalmente actividades de investigação;

Fase 2) Fornecer à Força Aérea know-how relativo à definição de requisitos técnicos e operacionais, à utilização e à operação de UAV's;

Fase 3) Criação de um projecto de I&T em conjunto com a Faculdade de Engenharia da Universidade do Porto, mais concretamente ao LSTS (Laboratório de Sistemas e Tecnologia Subaquática) para a criação e desenvolvimento de hardware e software para os UAV’s.

Este Programa desenrola-se, como atrás se disse, por três fases. Até agora foram concretizadas as duas primeiras. A terceira teve início em Janeiro de 2009, e prolongar-se-á até Dezembro de 2015.

Figura 2: UAV Antex-X02 e suas características operacionais.

Figura 3: UAV Antex-X03 e suas características operacionais.

Actualmente a equipa que está a desenvolver o sistema de software e hardware no LSTS possui já uma vasta bateria de testes que tem ajudado a desenvolver os protótipos até ao momento. Mas no entanto, está ainda deficitária de um sistema que possibilite quer a validação do sistema quer a formação dos operadores futuros do sistema.

1.3 Motivação e Objectivos

A cada dia que passa, cada vez mais o ser humano se vê rodeado de sistemas autónomos e veículos não tripulados. É notória a necessidade de uma plataforma de testes cada vez mais robusta e precisa. Esta necessidade provém de uma maior exigência de garantia de segurança por parte dos compradores, utilizadores e de todos os possíveis afectados por este sistemas.

Hoje em dia, estes sistemas estão em todo o lado e, cada vez mais, o ser humano as lida diariamente sem se aperceber das mesmas, mas são os sistemas de maior dimensão e mais complexos onde existe a maior possibilidade de ocorrência de erros graves e até de problemas com fins bastante trágicos.

Isto acontece pois quanto mais complexo se torna um sistema mais difícil é testar e certificar a sua segurança, e por vezes impossível determinar todos os possíveis desafios com que o sistema irá ter de lidar.

Neste âmbito o teste de integração de software torna-se fulcral, quer a um nível mais formal como também a um nível mais prático e sistematizado, como por exemplo a execução de testes por simulação. Desta forma é possível garantir uma maior fiabilidade dos sistemas, e também uma maior abrangência de reacções que o sistema terá de efectuar quando colocado em determinado ambiente.

Existe um conjunto de acidentes com sistemas autónomos que só vêm provar a necessidade que estes sistemas têm de uma maior abrangência de testes possíveis. Acontecimentos estes que por vezes colocaram a vida de dezenas e centenas de pessoas em risco e até mesmo possam ter causado a morte a algumas centenas pessoas. Como é do conhecimento geral, nos sistemas críticos não existe margem para erro.

Exemplo disso é o voo 124 das companhias aéreas da Malásia que fazia a ligação de Perth para Kuala Lumpur no dia 1 de Agosto de 2005. Mais ou menos 18 minutos depois da descolagem enquanto o Boeing 777-200 estava a passar os 10 972 m de altitude em piloto automático, este começou a subir 18 graus o nariz do avião, e começou a subir rapidamente sem qualquer aviso à tripulação. Quando o avião passou os 11 887 m de altitude o sistema começou a dar o aviso de stall e de velocidade excessiva, algo que não deveria ser possível e para o qual a tripulação não está treinada para lidar [7].

Aos 12 496 m de altitude o comandante desligou o piloto automático e desceu o nariz do avião, e então o acelerador automático deu a instrução de aumentar a potência dos motores, e o avião desceu 1219 m. Depois o piloto corrigiu o acelerador manualmente para a posição base, mas o avião voltou a levantar o nariz e a subir 609 m antes que o piloto conseguisse voltar a ganhar o controlo da aeronave.

A tripulação avisou o controlador aéreo que estava com dificuldades em manter a altitude e que iriam regressar ao aeroporto de partida. Felizmente, todas as 177 pessoas regressaram ilesas do incidente.

As autoridades canadianas estudaram o caso e descobriram que a unidade de inércia (ADIRU) que fornece informações sobre a velocidade, a altitude, o posicionamento e comportamento do aparelho tinha dois acelerómetros avariados. Um deles avariado desde 2001 e o outro avariou aquando o avião passava os 11 146 m de altitude.

A tolerância a falhas do sistema ADIRU tinha sido desenhado para operar com um acelerómetro avariado, visto que a aeronave tem seis. A grande redundância de desenho surge quando não era obrigatória a substituição do componente quando este avariava. E os mecânicos sempre pensaram que este ainda trabalharia com os restantes cinco sem problemas, mas o sistema não sabia como lidar com mais do que um acelerómetro avariado.

Então, quando o segundo sistema falhou, uma falha de software permitiu que os erros do primeiro sistema pudessem ser usados e assim resultar numa aceleração errada no sistema de piloto automático. Esta anomalia, que esteve escondida uma década, não foi encontrada durante os testes pois a equipa de desenvolvimento nunca considerou que tal erro pudesse acontecer.

“Existe sempre um conjunto de circunstancias que não são previstas, e para o qual o autómato não foi desenvolvido, ou simplesmente não eram situações esperadas,” diz Parasuraman. Sendo que à medida que a fiabilidade de um sistema chega perto, mas não nunca alcança, os 100% “cada vez é mais difícil é detectar e recuperar de um erro”. E quando o operador humano não consegue detectar o problema as consequências podem ser enormes [7].

Figura 4: Acidente no metropolitano de Washington. Foto: Win McNamee/Getty Images

Em Junho de 2009 ocorre outro dos casos que vem provar esta urgente necessidade. Um metropolitano de Washington conduzido por Jeanice McMillan colidiu com a traseira de um outro veículo estacionado fora da estação de Fort Totten a noroeste de Washington, matando a condutora e oito outros passageiros, causando ainda ferimentos em outros oitenta[7].

O acidente ainda se encontra em investigação por parte de autoridade americana para a segurança nos transportes (NTSB), mas tudo aponta para a falha de um sinal de segurança mal construído, que terá replicado um sinal de passagem quando esta seria proibida. Devido à locomotiva estar parada neste sinal avariado o sistema não conseguia detectar o comboio pensando assim que a linha estaria livre e transitável quando na verdade não estava. Assim que McMillan avistou a outra composição parada esta activou os travões de emergência mas não conseguindo parar a composição a tempo.

Em perspectiva, este foi o segundo acidente fatal em trinta e três anos de história do metro de Washington. Em 2008 os utilizadores realizaram 215 milhões de viagem no sistema metropolitano. E a automação trouxe grandes benesses quer para o sector dos comboios mas também para o sector da aviação. Segundo a Boing em 2000 viajaram de avião em todo o mundo 1.09 mil milhões de pessoas em 18 milhões de voos e apenas existiram 20 quedas fatais. E ainda o NTSB estima que as mortes na estrada iram cair cerca de 30% quando o controlo electrónico de estabilidade se tornar obrigatório na América em 2012.

Actualmente a crescente complexidade destes sistemas tem revelado uma preocupação mais elevada por parte das autoridades de certificação e de teste de software. E cada vez mais os desenhadores de sistemas estão a tentar automatizar o máximo possível as interacções complexas entre hardware e software. Muitas das vezes encontrando aspectos nunca antes previstos por parte dos mesmos na concepção dos modelos e que muitas das vezes os seres humanos não estão habituados a lidar.

"As coisas realmente difíceis de automatizar ou sintetizar, deixamos para o operador as fazer", diz Ericka Rovira, uma professora assistente do curso de engenharia psicológica na E.U. Academia Militar de West Point. Isso significa que as pessoas têm que estar alerta e pronto para agir nos momentos mais cruciais, mesmo que a monotonia dos sistemas vigiados os deixem fisicamente adormecidos ao volante[7].

Figura 5: Navio Royal Majesty encalhado ao largo de Massachusetts. Foto: Steven Senne/AP Photo

E isso foi o que causou o acidente em Junho de 1995, o navio de cruzeiro com 173 m de comprimento Royal Majesty ficou encalhado num banco de areia a 16 Km a leste da ilha de Nantucket na costa de Massachusetts. Cinquenta e dois minutos depois de ter partido do porto de St. George's, nas Bermudas, seguia em direcção a Boston quando o cabo da antena do GPS se desprendeu da antena. Isto fez com que o GPS passasse a trabalhar em dead-reckoning, que o faz calcular a posição consoante as posições anteriores, mas não tem em consideração o vento e as correntes marítimas. O GPS apesar de avariado continuou a dar valores ao piloto automático do navio e ninguém reparou no estado do GPS, apesar de este ter de ser verificado todas as horas comparando-o com o sistema de rádio Loran-C, que tem uma precisão de erro entre 800 m a 1 600 m em alto mar e 400 m perto da costa. O navio permaneceu em piloto automático durante 34 horas em dead-reckoning [7].

Porque é que os oficiais não notaram que estava algo errado? A maior razão segundo a NTSB foi o facto de os oficiais se terem tornado extra confiantes no sistema de segurança do navio.

Em muitos aspectos os operadores necessitam cada vez mais de serem administradores responsáveis e preparados para assumir o comando no meio de uma situação complexa, em que o sistema ou não foi desenhado, ou não está preparado para lidar com essa situação. Estes têm de ser rápidos a julgar e a diagnosticar o problema para depois descobrir uma solução satisfatória e segura. E se não o estão, são estes que acarretam com as culpas, não os desenvolvedores do sistema.

Criar um outro sistema para ajudar a detectar o erro e recuperar, encontrando uma solução segura, não é simples nem barato. No voo 124, a tolerância a falhas, ajudara unicamente a

mascarar o problema. Na verdade, a recuperação de falhas serve apenas para funcionar como mais uma camada que abstrai o operador humano do controle operacional do sistema.

"Por outras palavras, o ciclo inicial do controle é feito por um sistema, que depois tem um outro computador que está a fazer o backup desse sistema principal, e ainda outro a funcionar como backup do segundo," de acordo com Parasuraman. “Finalmente, é cada vez mais necessário formar o operador do sistema, mas no entanto o operador está cada vez mais distante de como o sistema realmente funciona, o que faz com que os seus mapas mentais sejam extremamente difíceis de construir, e por isso seja muito difícil ao operador lidar quando algo corre mal”.

Figura 6: Local de colisão do voo 1951 das linhas aéreas Turcas. Foto - United Photos/Reuters

Por exemplo hoje a degradação das capacidades dos pilotos é cada vez mais comum. Peter Ladkin, Professor de redes de computadores e sistemas distribuídos na Universidade de Bielefeld, na Alemanha, está muito ligado às investigações de acidentes de aviões, e ele próprio é um piloto. Este professor disse: “ Os pilotos hoje são treinados para usarem o piloto

automático a todo o momento. Muitos dos pilotos mais velhos estão preocupados que quando entrarem numa situação crítica, estes não saberão o que têm de fazer para sair dessa situação".

O acidente de Fevereiro das linhas aéreas Turcas, voo número 1951, perto do aeroporto de Amesterdão, Schiphol International Airport, matou nove pessoas e feriu 86. Este acidente realçou este problema. Quando o avião passava os 594 m de altitude o altímetro da asa esquerda falhou e começou a dar um valor de 2 metros negativos, então o piloto automático reduziu a aceleração dos motores pois supôs que estaria na fase final da aterragem. Os pilotos não reagiram inicialmente aos avisos de que algo estava errado até ser demasiada tarde para recuperar o avião[7].

"Quando começamos a tirar a aprendizagem activa do operador, este começa a confiar em demasia no autómato," diz Rovira. "Eles não poderão voltar atrás e recuperar toda a informação" para tomar uma decisão correcta.

Estes são exemplos claros que, cada vez mais, a validação e o treino de operadores são importantes sendo mesmo fulcrais no caso de UAV's. Estes são dispositivos a funcionar a uma distância considerável e cuja queda pode causar prejuízos elevados, tornando-o mesmo um sistema crítico em que, se algo falhar numa missão extremamente sensível pode comprometer a vida de civis e militares.

Esta dissertação tem como principal objectivo desenvolver uma ferramenta que permita endereçar os 4 tipos de falhas referidas, sendo elas:

- A existência de cenários nunca antes previstos no contexto do projecto e por isso o avião não terá sido preparado para esse caso.
- A prevenção e resolução autónoma de erros de sensores e de componentes e sistemas do avião ou groundstation.
- O excesso de confiança por parte dos operadores perante um sistemas que pode falhar a qualquer momento.
- Finalmente a possibilidade de em caso de erro o operador do sistema ser incapaz de tomar as medidas correctas para recuperar o avião trazendo-o de volta para a base em segurança.

Sendo assim o projecto abrange pontos muito importantes, por um lado o teste e validação dos sistemas integrados no UAV, e por outro lado a possibilidade de passar a existir uma plataforma onde é possível realizar ensaios e treino de operadores para missões reais no terreno.

Desta maneira é possível colmatar muitas das falhas a diversos níveis, como por exemplo, erros de desenho, erros de implementação, situações inesperadas na concepção do sistema e ainda e não tão pouco importante a formação dos utilizadores para reagirem o mais rápido e eficientemente possível aos erros e problemas que se iram deparar em pleno ar.

1.4 Estrutura da Dissertação

Para além deste capítulo, a introdução, esta dissertação está dividida em mais 6 capítulos. Seguidamente será apresentado o estado da arte onde alguns projectos associados ao teste por simulação serão demonstrados. No capítulo 3 será apresentada uma análise do problema onde será explicado o método a abordar para a resolução do problema. No capítulo 4º serão apresentadas as formas de implementar a solução bem como um conjunto de casos de uso que foram acordados com o laboratório e o orientador. No 5º capítulo serão apresentadas as conclusões e as perspectivas de trabalho futuro. Finalmente no 6 capítulo serão apresentados os anexos onde estarão os diagramas realizados para a análise de mensagens.

2 Revisão Bibliográfica

2.1 Introdução

Este capítulo tem como objectivo a demonstração de actuais projectos em curso bem como outros já realizados no âmbito de testes por simulação e formação de utilizadores. Apesar da existência de poucos exemplos, isto não quer dizer que as empresas não apostam neste tipo de tecnologias para validar o seu software e formar utilizadores, muito pelo contrário, isto quer dizer que actualmente esta é uma área cada vez mais sensível nas empresas e por isso muitas das vezes os sistemas são secretos e fechados dentro das empresas. Isto porque estes sistemas estão construídos para encontrar falhas nos seus produtos e por isso estas empresas não querem que o seu próprio motor de testes seja motivo de uma falha no seu sistema. Por este motivo o conjunto de sistemas conhecidos não são muito elevados, e por isso só serão apresentados um conjunto restrito de projectos e metodologias.

2.2 Actualidade

Nesta secção será apresentado um conjunto de projectos já realizados nesta área bem como os seus pontos fortes e fracos em análise. Grande parte destes projectos são de áreas distintas da aeronáutica mas que usam o mesmo paradigma de teste por simulação quer para teste de software quer para treino de equipas para operar estes sistemas.

2.2.1 Testes de Simulação

Actualmente o American Institute of Aeronautics Astronautics (AIAA) está a desenvolver um conjunto de padrões, o FDMES a ser integrado no conjunto de padrões da American National Standards Institute (ANSI), para a elaboração de simulações. Estes padrões têm por base a partilha de informação entre uma comunidade que executa testes por simulação deste tipo. Desta forma os sistemas possuem uma vasta e ampla gama de simulações já realizadas, tornando assim a combinação mais simples. Esta instituição afirma que os ganhos deste conjunto de padrões são vastos, sendo eles:

- Os modelos aerodinâmicos e as suas funções passam a ser fáceis de manter actualizados pois são partilhados por desenvolvedores, simuladores de voo, e simuladores de treino de todo o tipo de arquitecturas.
- Deixa de ser necessário reescrever e alojar as funções e de grande parte dos modelos aéreos.
- Ficheiros de resultados fáceis de interpretar por sistemas e humanos.
- O padrão evita linguagens proprietárias, standards e práticas.
- O padrão é facilmente expansível para acomodar novas características.
- O padrão é versátil e fácil de mudar.
- Existe um aumento da documentação sobre todos os acontecimentos.
- Baixo custo e de tempo que demora a produzir a simulação.
- Criação de modelos mais fiáveis.

Até ao momento da escrita desta dissertação a ANSI está aberta a propostas e comentários de pessoas ligadas à área.

2.2.2 National Aeronautics and Space Administration (NASA)

A NASA (National Aeronautics and Space Administration) desde muito cedo usa simuladores quer para teste de sistemas quer para treino de astronautas. Estes prestam um importante papel quer para manter os astronautas na máxima segurança e principalmente treiná-los para o máximo de situações possíveis para que, estes saibam exactamente o que executar numa situação de emergência.

Apesar de a NASA investir bastante nesta área existem sempre cenários não previstos. É o caso da missão lunar Apollo 13. Esta foi lançada no dia 11 de Abril de 1970 do Cabo Kennedy com a missão de aterrar na Lua. Durante os primeiros dois dias da missão, a viagem correu tranquilamente. As nove horas da noite do dia 13 de Abril e com 55 horas e 55 minutos de missão, os três astronautas ouviram e sentiram um grande estrondo nas entranhas da nave. No dia 17 de Abril de 1970 estes homens voltaram a terra sem nenhuma perda humana no acidente[8].

Figura 7: Exemplo de plataforma de testes no laboratório da NASA. Reconstituição feita no filme Apollo 13.

A NASA durante todo o processo usou um simulador para treinar e para testar situações em que os pilotos eram colocados em situações extremas e tinham de responder de maneira rápida e o melhor possível. Mesmo depois do acidente, em terra, os técnicos não pararam de fazer simulações para verificarem a viabilidade da colocação dos sistemas em certa configuração.

A NASA usa este tipo de sistemas convencionalmente para teste e aperfeiçoamento dos operadores quer terrestres, quer os astronautas. A mesma instituição usa uma abordagem semelhante no caso dos controladores terrestres em que são colocados em situações críticas e têm que reagir de igual maneira que os astronautas, pois muitas vezes a sua vida dependia do sucesso dos sistemas terrestres.

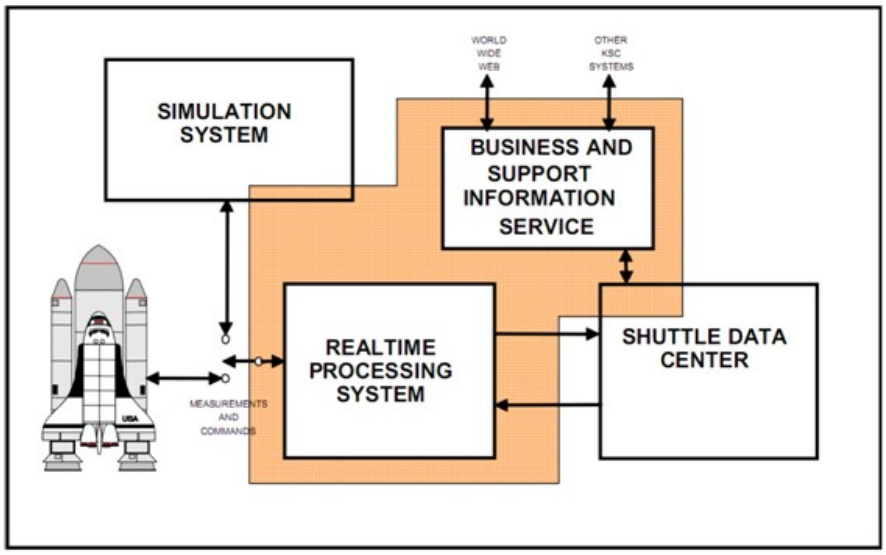

Figura 8: Arquitectura do sistema de lançamento e teste. Retirado do NASA Shuttle ground operations simulator (SGOS)[3]

Sendo assim a NASA realiza quatro tipos de simulação em grupo para os seus colaboradores, sendo elas:

1. **Treino diferenciado para a equipa de lançamento.**

Serve para teste e treino de equipas e sistemas para o lançamento das aeronaves.

Este é dividido por três níveis distintos, o terceiro nível, o mais alto, destina-se a testar sistemas completos como por exemplo os testes antes de levantar voo. O segundo nível serve para testar sistemas que ainda possuam outros sistemas interligados e, finalmente, o último nível serve para teste de componentes sem que estes dependam de outros sistemas.

2. **Simulação de treino para o voo de equipas de astronautas e para equipas de controladores de missão.**

Este treino é concebido para os astronautas e para todos os operadores da sala de controlo de missão.

3. **Simulação de treino para a equipa de gestão de missão.**

Estes são dos elementos mais importantes de uma sala de controlo de uma missão. São estas equipas que tomam todas as decisões mais cruciais durante uma missão.

4. **Simulação e treino para os veículos da NASA.**

Esta equipa tem o dever de testar e simular o comportamento dos veículos fora da órbita terrestre. Estes têm um papel fulcral no teste e simulação dos veículos não tripulados.

Figura 9: Spirit, mission designation MER-A (Mars Exploration Rover - A)

2.2.3 EADS-Astrium SIMSTB(SIMBVL)

Segundo o estudo de C. H. Koo e S. K. Lee [5] existem vários tipos e fases de validação de software de voo. Como por exemplo testes unitários, testes de integração, testes de sistemas, entre outros. Estas ferramentas e métodos servem como ferramentas feitas para os desenhadores as usarem em pequenos testes.

Como é do conhecimento geral os testes de integração têm como propósito a validação da compatibilidade entre hardware e software, assim sendo a plataforma onde o software vai ser executado é essencial para este teste extremamente vital. Para uma validação correcta é necessário que o sistema esteja a funcionar como um todo tal como executará na realidade.

Mas no entanto correr em plataformas computorizadas tem algumas desvantagens. Por exemplo, o tempo que demora a operar estes testes é demasiado elevado e estes sistemas não podem ser executados por mais do que um tester, sendo assim o teste torna-se ineficiente, caro, frágil e complicado pois unicamente oferece uma possibilidade de um para um.

Hoje em dia o desenvolvimento de software para satélites é um grande desafio, devido aos períodos cada vez mais reduzidos e normalmente ao desenvolvimento de múltiplas soluções sobre a mesma plataforma. Assim sendo os métodos de validação em satélites têm de ser extremamente eficazes.

EADS-Astrium usa vários tipos de simuladores durante a validação de algoritmos, testes de sistemas e validação das operações do satélite, isto é validação de software. Estes simuladores são resultado de uma intensa investigação nas Series E3000 e seus produtos relacionados a nível de algoritmos e operacionalmente [5].

EADS-Astrium também usa vários tipos de simuladores de ambientes para desenvolvimento e validação dos sistemas de voo. Um dos sistemas é chamado de SIMSTB(SIMBVL). Sendo assim os sistemas de voo dos satélites podem ser validados usando o SIMSTB durante os testes de integração.

As vantagens de utilização do SIMSTB para a avaliação são:

- Fácil de configurar e usar;
- Flexível;
- Multi-utilizador
- Teste automatizado graças ao test script
- Tudo agrupado no mesmo simulador, setup e modelo de voo.

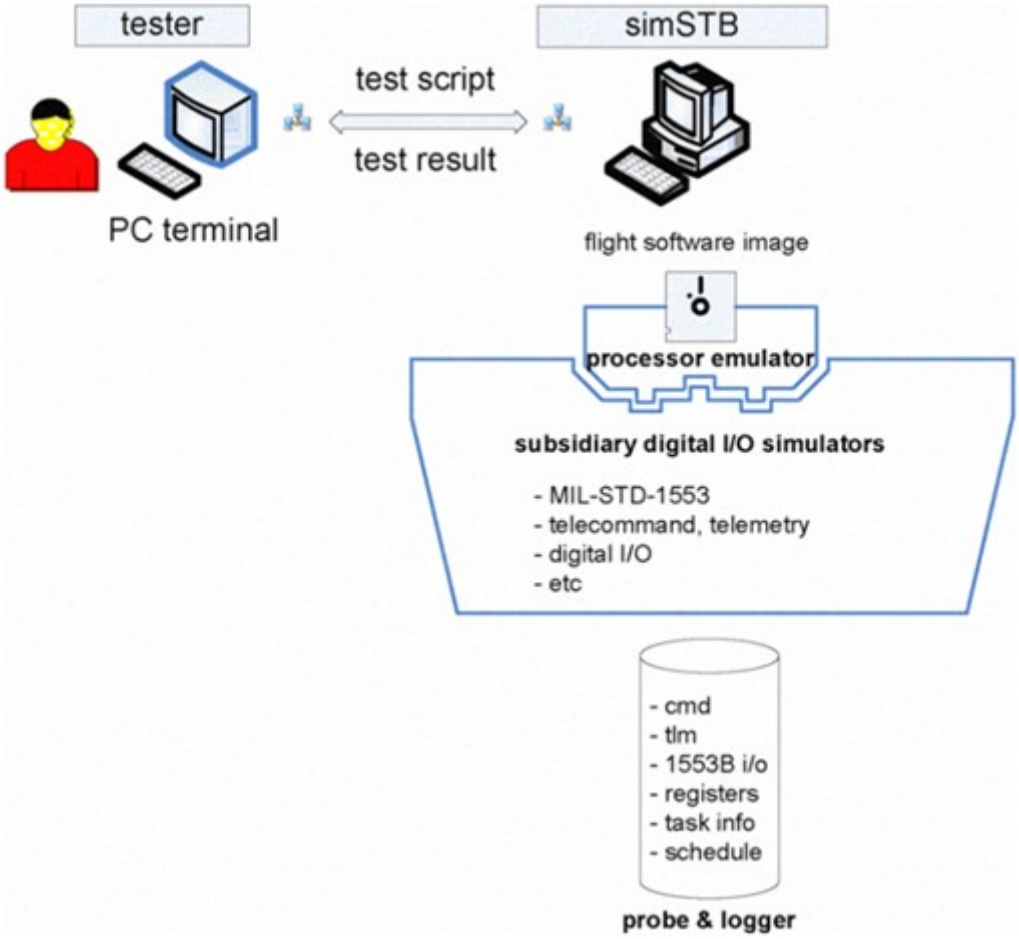

Figura 10: Configuração básica do simSTB. Retirado do artigo Approach to the use of simulated Software Test Bench in integration test of flight software [5]

2.2.4 MATLAB Automated Testing Tool (MATT)

Segundo esta ferramenta o teste de sistemas em tempo real apresenta um conjunto de desafios bastante interessantes. O teste destes sistemas muitas vezes tem de ser feito sem colocar o sistema de hardware em off-line ou não pode ser separado o que pode provocar danos nos sistemas e mesmo custos elevados. No caso de aplicações aeroespaciais, por exemplo, muitas das vezes o hardware nem sequer existe quando alguns dos componentes precisam de ser testados. Muitos dos testes são até perigosos de se realizar, como por exemplo em software de controlo de mísseis, enquanto não existir um nível de segurança bastante elevado visto que o risco de danos elevados é bastante alto.

As simulações apresentam vários problemas como a potencial geração de quantidades enormes de dados, por exemplo, se um sistema recebe 500 estímulos em 100 milissegundos, este terá de introduzir 5 000 valores por segundo. Uma modesta simulação de uma hora pode requerer 18 000 000 de valores. Se todos estes valores forem guardados em valores de vírgula flutuante equivalem a ocupar 72 megabytes de informação inserida, e se o sistema responder com a mesma velocidade, equivale a mais 72 megabytes de informação recebida. A análise e cruzamento desta informação torna-se assim um trabalho massivo. Mas no entanto imaginemos um sistema que tem de ser testado durante meses e anos como seria.

Outro problema, bastante mais sério, é que o teste por simulação pode não cobrir todas as condições de entrada e todas as situações que poderão ocorrer na realidade quando este estiver em plena função. Quando isto acontece e o sistema entra numa situação crítica, os prejuízos podem ser bastante elevados.

Assim para colmatar esta falha pode ser usada uma metodologia baseada em conjuntos de funções, como testes tipo, que são usadas para criar conjuntos de valores de entrada que podem ser configurados para cobrir o perfil operacional do sistema em tempo real. Para isto os procedimentos são implementados usando uma ferramenta de teste, MATLAB Automated Testing Tool (MATT), que torna autónomo o processo de geração de casos de teste, estados de simulação, teste ao código fonte e análise de resultados[2].

Neste artigo [3] chegou-se à conclusão que os tipos de teste para as aeronaves, em geral, seriam:

- Controlar o comportamento do dispositivo no campo.
- Transição de um componente ou subsistema de estável para instável.
- Transição de um componente ou subsistema de instável para estável.
- Testar de maneira a aproximar-se dos valores de entrada críticos.

Estes testes tipo não são feitos para substituir os testes específicos a cada componente mas sim para complementá-los. Por outras palavras os engenheiros de sistemas utilizam o seu conhecimento para prever o comportamento esperado e configurar os testes focando-se num requisito específico. Neste caso não, os engenheiros exploram a capacidade do software devolver o output específico.

Como podemos observar nas imagens seguintes à medida que testes vão sendo executados e corrigidos a fiabilidade de um sistema aumenta drasticamente. Por isso é de extrema importância o uso destas técnicas para o teste de software. Isto porque a longo prazo o número de falhas diminui drasticamente (figura 11 e 12).

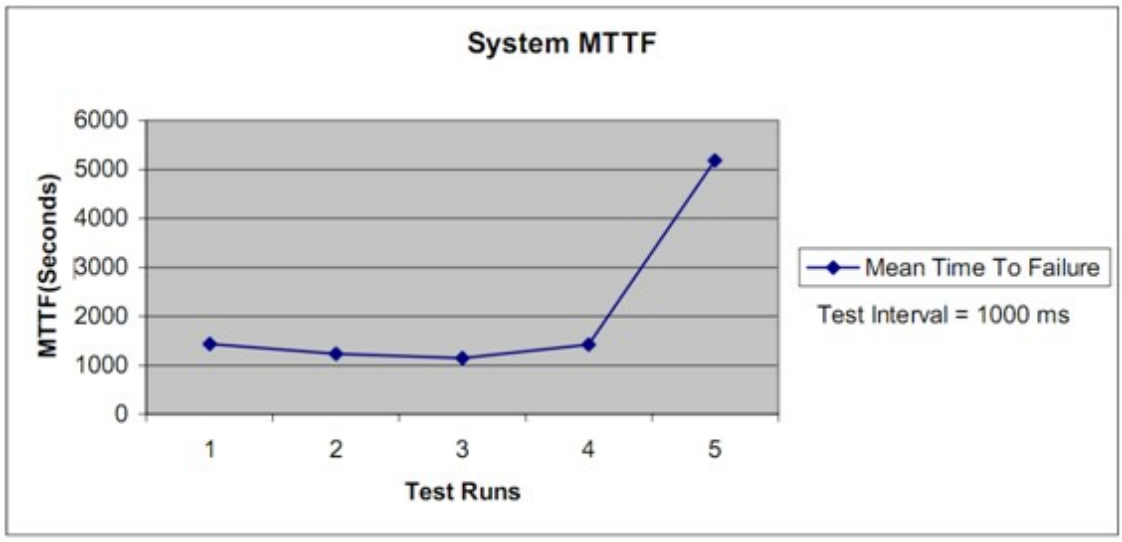

Figura 11: Tempo médio de operação até falhar consoante o número de testes realizados.

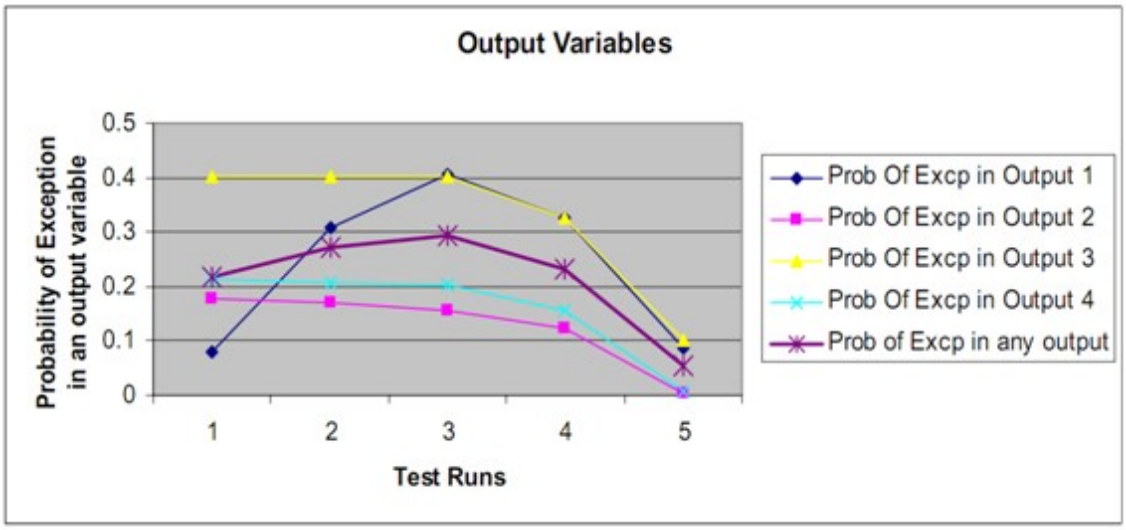

Figura 12: Probabilidade de ocorrência de erros ao consoante o número de testes realizados.

2.3 Sistema e ferramentas

Nesta secção serão apresentados todos os sistemas e como actualmente interagem. Isto porque um avião é composto por vários tipos de sistemas e ferramentas, que todos juntos formam o sistema de voo. Assim sendo um avião é constituído por três sistemas e um protocolo de comunicação que serve de interacção com a terra.

Assim sendo, o sistema é composto por:

- Neptus – Software de comando em terra;
- Dune – Software de gestão do veículo no ar;
- Piccolo – Piloto automático do veículo no ar;
- IMC – Protocolo de comunicação terra – ar.

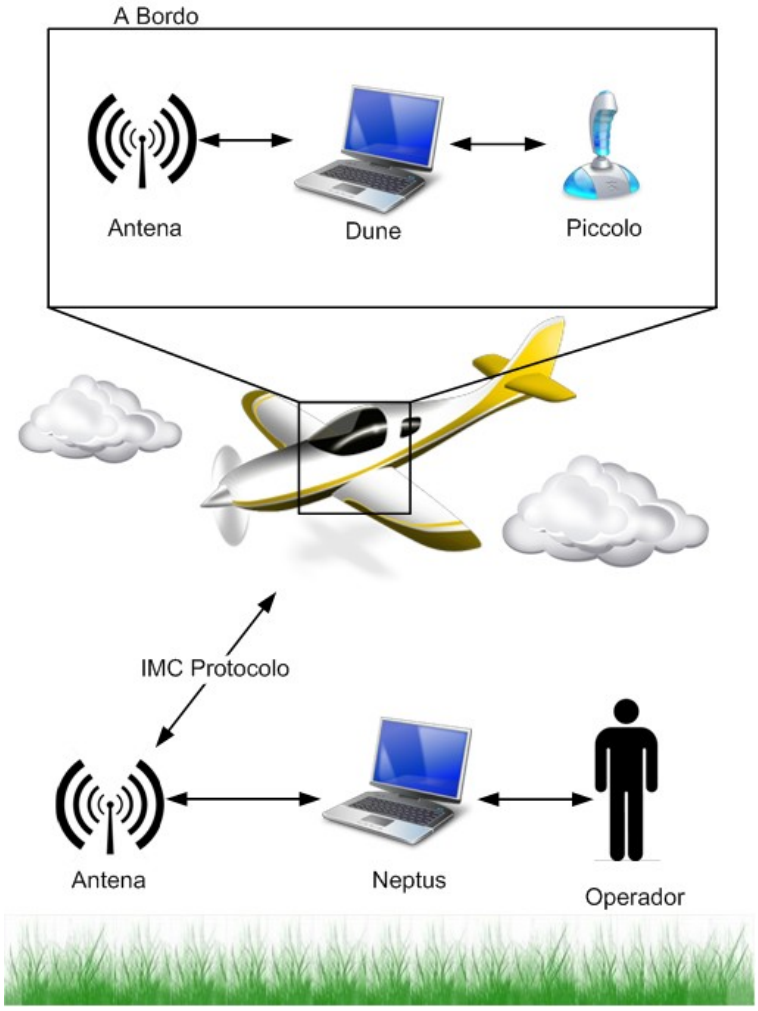

Figura 13: Diagrama de interacção dos sistemas de voo.

Os sistemas interagem como o representado na figura seguinte e será seguidamente explicado cada componente para uma mais fácil percepção.

De salientar que este não é o real set up de voo pois actualmente à estação terrestre está ainda ligado um piloto com a capacidade de tomar controlo do aparelho quando necessário. O motivo por não estar referenciado é somente porque este é o set up que se pretende implementar futuramente, onde o trabalho do piloto é feito pelo operador de voo.

2.3.1 Netpus

Neptus [9] é o programa de comando e controlo, desenvolvido pelo LSTS, que está a correr na Ground Station (estação terrestre) e serve de interface entre o operador e o avião. Este é uma Framework que permite controlar equipas de veículos, quer de sejam do mesmo tipo ou não. O Neptus permite a interacção com vários tipos de veículos, quer sejam autónomos, semi-autónomos ou teleguiados.

O sistema permite definir uma missão ou missões consoante se esteja a operar um ou vários veículos, e estes consistem em vários mapas e planos individuais para cada um deles. Quando em equipa, os veículos para além de comunicarem com a Ground Station através do protocolo IMC também o fazem entre eles através do mesmo protocolo.

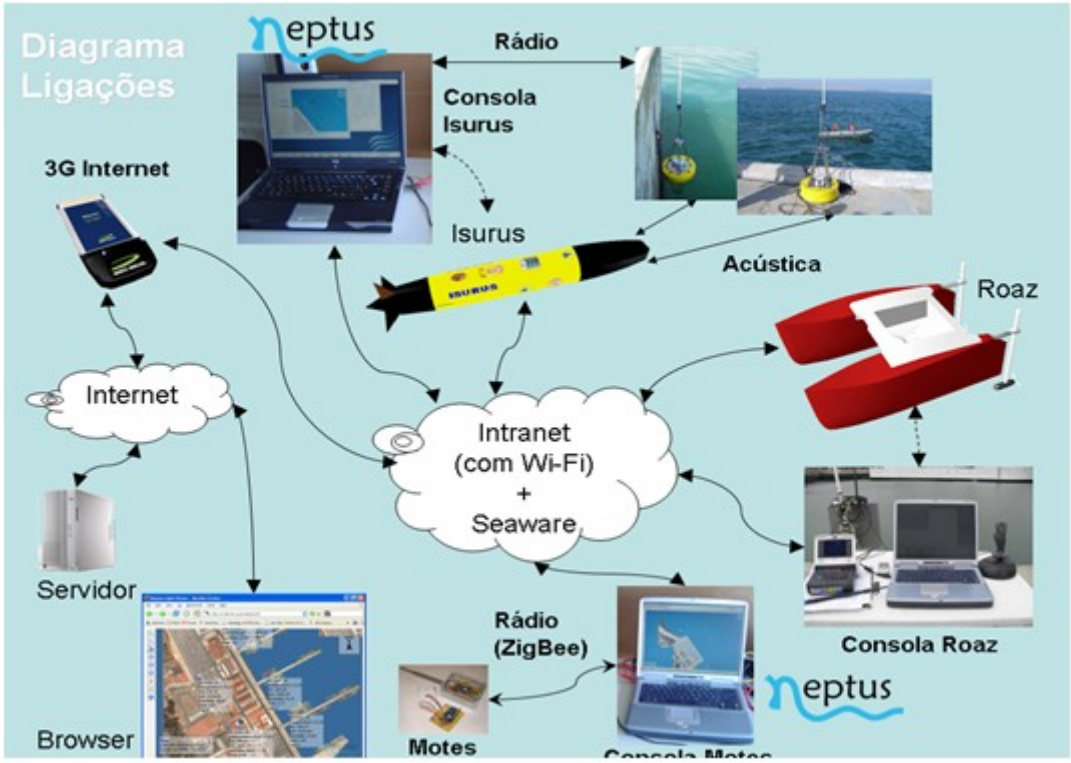

Figura 14: Nuvem de comunicações do Netpus.

A Ground Station é constituída por um computador em que neste corre o programa Neptus e ao qual a ele está ligado um transmissor e receptor rádio para poder comunicar com os veículos sem a necessidade de cabos e outros acessórios.

O sistema foi inicialmente criado para operar com os submarinos do laboratório mas actualmente já possui inúmeras funcionalidades para operação com UAV's. O objectivo é que um operador consiga ter todos os veículos abrangidos por este sistema a operar ao mesmo tempo e a serem monitorizados por um único comando de operações, fazendo assim uma mais fácil coordenação dos meios envolvidos.

O sistema é bastante flexível na forma de ser facilmente possível adicionar novos tipos de manobras e veículos. Para além disso podem ser facilmente criadas consolas de operação para novos tipos de veículos ou missões. Tornando-o assim bastante personalizável para satisfazer os gostos quer de Flight Manager ou operador de voo, pois podem dispor a informação no ecrã como melhor desejarem [10].

Por exemplo na imagem seguinte é mostrada uma janela de comando de um avião, onde podemos retirar informações da posição, altitude, velocidade e também receber o output de vídeo que está a ser gravado no avião. Neste caso, o teste foi feito sem câmara de vídeo e por isso a imagem está a preto, e esse é o motivo da existência do quadrado preto à direita.

Figura 15: Exemplo de uma consola Neptus de comando.

2.3.2 IMC

IMC, ou Inter-Module Communication protocol, foi desenhado e implementado pelo LSTS e é um protocolo de comunicação preparado para veículos autónomos. Estes podem ser terrestres, marítimos ou aéreos. O protocolo está preparado para trabalhar com sensores, com operadores humanos e também é compatível com standards internacionais, exemplo disso é o STANAG 4586. O controlo dos veículos pode ser autónoma ou semi-autónoma, como por exemplo teleguiado.

Um dos principais objectivos deste projecto foi criar uma plataforma que todos os veículos usassem para que seja possível a comunicação entre os mesmos, e ainda uma mais fácil comunicação entre uma estação terrestre e os veículos em missão. Desta maneira é possível um veiculo comunicar com vários tipos de estações terrestres e até mesmo ser operado por diferentes pessoas no espaço de muito pouco tempo. Existe ainda a capacidade de se integrar completamente com o Neptus o que faz com que nesta plataforma seja possível coordenar vários veículos em simultâneo.

O IMC tem o seguinte conjunto de tipos de mensagem:

- Mensagens de controlo de missões;
- Mensagens de controlo de veículos;
- Mensagens de manobras;
- Mensagens de orientação;
- Mensagens de navegação;
- Mensagens de sensores;
- Mensagens de actuadores.

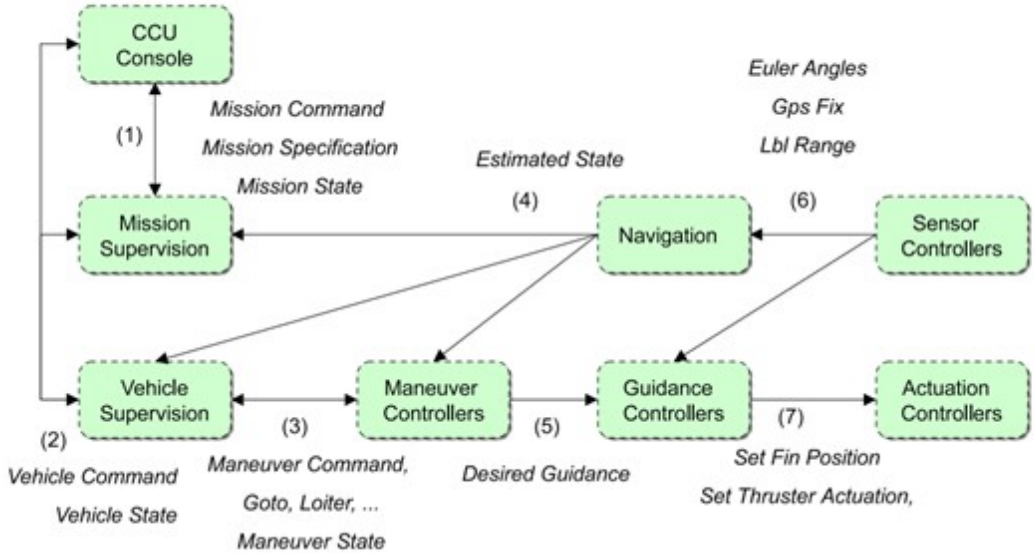

Figura 16: Fluxo de mensagens no submarino Seascout.

2.3.3 Dune

O Dune é o sistema desenvolvido no LSTS e é uma das aplicações que vai a bordo do avião, este corre num pequeno computador que tem uma capacidade de processamento mais reduzida devido ao seu diminuto tamanho. Este computador é chamado de Pc-104 e corre o sistema operativo Linux. Esta ferramenta é responsável por comandar o UAV e contém drivers para sistemas de aquisição de informação, navegação, controlos de manobras e para simulação com hardware-in-the-loop.

Este sistema corre numa Compact Flash que vai ligada à Pc-104 e uma vez o computador ligado este programa arranca automaticamente. Uma vez em voo o Dune está responsável por comandar o Piccolo e gerir a comunicação com terra. Existe ainda a possibilidade de ligação de vários dispositivos como de aquisição de vídeo, radares e vários outros sensores.

Actualmente, o Dune está já a funcionar habitualmente ligado a uma câmara usb para captação de vídeo em voo e também como comandante de voo para o piccolo.

A nível de comunicação como Piccolo este está responsável por receber ordens de terra e através dessas enviar ordens de voo para o Piccolo as executar. Actualmente ainda só redirecciona as ordens de terra para o piloto automático mas futuramente espera-se que por exemplo este dê ordens, em tempo real, de qual deverá ser a posição do avião para uma mais clara imagem de vídeo ou por exemplo dar ordens de como seguir um veículo através dos dados recebidos pela câmara de vídeo.

O Dune comunica através do protocolo IMC com terra e usa o standard de mensagens da CloudCap para comunicar com o Piccolo.

Figura 17: Frente de uma Pc-104

Figura 18: Traseira de uma Pc-104

2.3.4 Piccolo

Piccolo é um sistema de controlo completo e integrado para pequenos aviões não tripulados vendido e produzido pela empresa CloudCap. O sistema é composto por vários sistemas de hardware e software. Este sistema contém a capacidade de cálculo de todas as variáveis físicas de um avião e também de cruzá-las com as informações adquiridas dos sensores internos. O Piccolo possui um GPS, um sistema de rádio e entradas para o tubo de Pitot e para o tubo de ar static [1].

Figura 19: Fotografia de um Piccolo.

Assim sendo o Piccolo acaba por ser um sistema autónomo de pilotagem de avião, a ele estão ligados todos os actuadores do avião e, através do modelo 3D do avião, este consegue pilotar o avião de forma completamente segura e estável.

Ao Piccolo também é possível ligar via rádio a uma Ground Station que corre um programa próprio de controlo desenvolvido pela empresa que comercializa o sistema, este programa possibilita o comando e acompanhamento da missão tal como o Neptus. Possibilita ainda a capacidade de criar simulações de voo injectando os valores físicos directamente no Piccolo fazendo-o pensar que está em voo real.

O Piccolo possui as seguintes capacidades de fábrica:

- Medidas do posicionamento do avião, deslocação e aceleração 3 eixos;
- Medições da pressão dinâmica e estática;
- Possibilidade de voo sem comando;
- Possui a capacidade de ligar a um computador externo para extensão de capacidades;
- Integração completa para sensores, datalink's e para servos;
- Vasta gama de suporte para software, incluindo a possibilidade de recepção de controlos externos;
- Software de comando para a Ground Station e simulação de voo;
- Sistema de pilotagem em tempo real completamente autónomo;
- Capacidade de ser montado em vários sentidos no avião;
- Diferentes opções de interfaces de payload.

2.4 Resumo e Conclusões

Em resumo, pode-se dizer que este é um sistema que por definição tem uma integração difícil, visto que é um conjunto de 3 sistemas independentes e em que cada um deles apresenta um conjunto de desafios únicos. Assim sendo o sistema de voo é constituído pelo Neptus para a interface com o operador, o Dune para lidar com os dispositivos a bordo e o Piccolo para lidar com toda a pilotagem automática do avião.

Também se pode reter que os testes por simulação têm bastante rentabilidade para a detecção de falhas e erros no processo de criação do sistema, e por isso é extremamente importante realizar este tipo de testes.

Grandes empresas têm este tipo de filosofias bem entranhadas nos seus processos e com óptimos resultados e por isso faz todo o sentido o laboratório faze-lo também.

3 Análise do Problema

Neste capítulo será analisado o problema, começando por falar um pouco do sistema actual e depois avançando para o real objectivo da dissertação. Inicialmente será falado de como o sistema actual funciona e depois falar-se-á do tipo de testes que serão realizados bem como qual o objectivo de cada um deles.

Será também descrito o conjunto de falhas que o programa irá tratar e tentar trabalhar de forma a que se tornem possíveis estes testes. De notar que estas falhas foram encontradas através do estudo de todas as comunicações entre os três sistemas.

3.1 UAS e validação por segregação

UAS que significa Unmanned Aircraft System, em português sistema de aviação não tripulado. O nome foi introduzido pelo Departamento do Defesa norte-americano e depois adoptado pela Administração federal de aviação (FAA). Um sistema destes consiste num conjunto de sistemas como um UAV, um sistema de controlo, um sistema de informação e o restante equipamento de suporte, como por exemplo catapultas.

No caso deste projecto temos um conjunto de plataformas que permitem o controlo e comando de cada missão. O sistema de voo actual é constituído por um piloto em terra que dá ordens à aeronave. Este operador tem a possibilidade de assumir o controlo a qualquer momento bastando para isso unicamente interagir com o sistema de comando. A aeronave está em contacto permanente com o meio ambiente, e é esta que envia a informação ao operador.

Figura 20: Princípio de interacção do sistema actual

O objectivo principal deste projecto é a junção de um novo sistema para a possibilidade de teste em simulação acrescida da interacção do programa juntando a capacidade de criar situações de falhas.

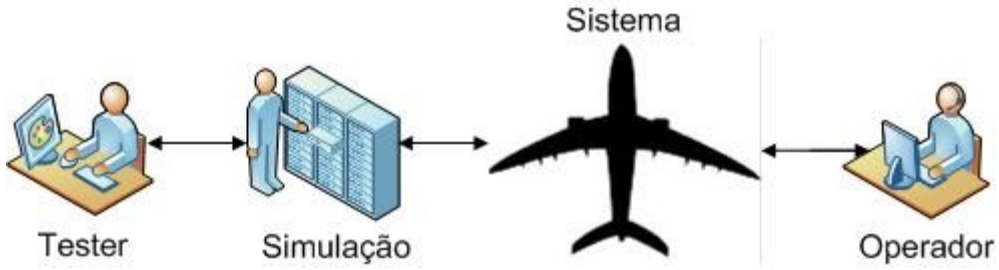

Figura 21: Interacção entre sistema, simulação, tester e operador.

3.2 Testes

Neste capítulo será abordado um conjunto de testes que serão realizados neste projecto. Será ainda apresentada a sua definição e objectivos para que seja mais fácil a percepção geral do projecto.

3.2.1 Testes de caixa Negra [11]

Testes de caixa negra são também chamados de testes funcionais, orientados aos dados de entradas e saídas dos sistemas. Estes têm como principal funcionalidade a avaliação externa do sistema sem ser considerado o seu comportamento interno. No caso deste projecto serão fornecidos dados de entrada (as mensagens) e será esperado um conjunto de outputs por parte

do sistema. A aprovação ou reprovação do teste é feito através de comparação com o resultado esperado para um dado input.

Quanto mais entradas são fornecidas, mais rico é o teste, idealmente todas as entradas possíveis deveriam ser testadas, mas infelizmente é virtualmente impossível cobrir todos os casos possíveis de uso e de entradas. Para além deste problema, estes testes exigem uma boa especificação dos sistemas em teste o que, caso não seja verdade, pode levar a testes inúteis e a erros que não são detectados por não estarem especificados antes dos testes.

Uma abordagem no desenvolvimento do teste de caixa negra é o teste baseado na especificação, por isso este documento ser tão importante, de forma que as funcionalidades sejam testadas de acordo com os requisitos.

3.2.2 Testes de integração [11]

Testes de integração são um tipo de testes realizado aquando a agregação de partes mais pequenas de funções ou sistemas, isto é este teste tem a responsabilidade de verificar se todos os sistemas e funções estão a obter os outputs desejados.

Para uma melhor garantia por partes destes testes convêm anteriormente já terem sido realizados testes aos módulos individuais para que a garantia de qualidade seja muito maior. Estes testes são realizados normalmente antes dos testes de sistema.

O propósito dos testes de integração é verificar os requisitos funcionais, de desempenho e de fiabilidade na modelagem do sistema. Com ele é possível descobrir erros de interface entre os componentes do sistema.

3.2.3 Testes de Sistema [11]

Estes testes são realizados normalmente logo depois dos testes de integração e têm como objectivo o teste de software e hardware em conjunto e com o sistema completamente integrado, neste caso será a realização dos testes com uma ground station e um Piccolo ligado a uma Pc-104.

Neste teste não é necessário conhecer a componente interna de cada sistema para realizar o teste mas só é necessário conhecer os dados de output esperados. É um teste mais limitado do que os testes de unitários e de integração, fases normalmente anteriores a este processo de teste, pois neste teste importam apenas aspectos gerais do sistema.

3.2.4 Testes não funcionais [11]

Testes de performance e capacidade

Os testes de performance são executados para determinar o quão rápido é um sistema ou sub – sistema debaixo de uma carga acrescida de trabalho. Serve também para validar e verificar outros atributos de qualidade do sistema como por exemplo a confiança, e a resistência ao stress. Testes de capacidade são realizados para determinar até que ponto o sistema é capaz de ser operável debaixo de stress constante.

Volume test são uma forma de testar a funcionalidade do sistema debaixo de elevadas quantidades de informação. Stress Test são uma forma de testar a confiança do sistema. Load test são uma forma de testar a performance.

Neste projecto é possível realizar este tipo de testes pois podemos variar o numero de mensagens que são trocadas, quer de forma ascendente ou descendente, isto é quer por envio de mais mensagens quer por envio de menos mensagens. Assim pode-se testar a quantidade de mensagens mínimas que são necessárias para operar e a quantidade máxima de mensagens que o sistema aguenta antes de entrar em estado de erro.

Testes de estabilidade

Testes de estabilidade têm como missão saber quanto tempo consegue o sistema consegue aguentar ligado e operar sempre nos parâmetros desejáveis. Este tipo de testes é normalmente referido como Load Test ou testes de resistência.

Testes de Destrutivos

Testes destrutivos tem como objectivo testar se o sistema aguenta quando este é deliberadamente colocado continuamente em situações de erros sucessivos, neste projecto refere-se à contínua modificação de mensagens para que se teste se o sistema é capaz de aguentar erros deliberados nas mensagens.

3.3 Falhas de sistema

Neste capítulo serão apresentadas os tipos de falhas de comunicação que se pretendem simular com projecto. Todos os subcapítulos estão acompanhados por uma ilustração para mais fácil interpretação e assimilação das falhas. De notar que as falhas criadas só estão relacionadas com os protocolos de comunicação entre os sistemas.

Estas são as falhas que se pretendem estudar e testar com este projecto. Estas foram determinadas depois de um estudo intensivo das comunicações entre os diversos sistemas.

3.3.1 Falha por bloqueio de mensagens de 1 para 2

Esta falha serve para simular uma impossibilidade de comunicação da entidade 1 para a entidade 2 isto é, a perda de comunicação é uni direccional sendo apenas bloqueadas as mensagens da entidade 1. Sendo que o sistema 2 terá de tomar por iniciativa própria o comando por exemplo se for o Piccolo. Este tipo de falhas tem por principal objectivo o teste da integridade do sistema e a formação de operadores visto que, o sistema entrará num estado em que a mensagem pretendida não passará sendo que o operador terá de encontrar maneiras de, ou introduzir novas ordens, ou mandar o avião para o lostcom point.

Figura 22: Bloqueio de mensagens de 1 para 2.

3.3.2 Falha por perda de comunicação

A perda de comunicação é uma simulação de perda integral de pacotes dessa mensagem, isto quer dizer que as mensagens desse tipo não circulam nem num sentido nem no outro, desta forma é possível testar a robustez do sistema para que caso exista uma perda de comunicação no ar. A nível de formação o sistema será assim capaz de preparar toda a equipa de operação para situações de perda de comunicações e de falhas sistemáticas no envio de ordens.

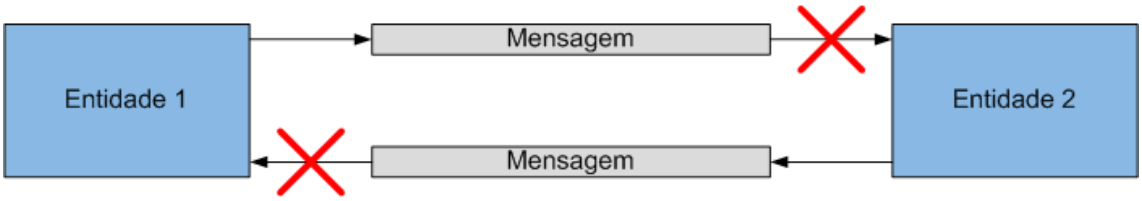

Figura 23: Perda de comunicação.

3.3.3 Falha por injecção de mensagens

Este é um tipo de falha que consiste na introdução de mensagem novas e sem lógica para a execução do sistema. Esta falha serve para testar por exemplo envio de mensagens de telemetria em que os dados são completamente aleatórios, em que o sistema terá de descartar esta mensagem e em alguns casos pedir de novo a mensagem à entidade 1. No entanto esta falha também serve para activar e desactivar opções de controlo do avião em treino de operadores, desta maneira o operador tem de ser capaz de detectar o que está errado e corrigir o erro.

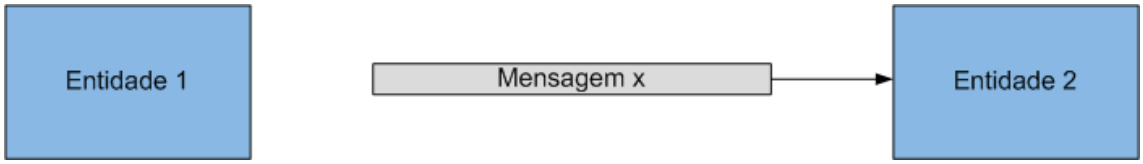

Figura 24: Injecção de mensagens.

3.3.4 Falha por recepção de feedback de mensagem desconhecida

Este tipo de falhas ocorrem quando são inseridas mensagens no sistema e a mensagem produz um feedback à entidade 1 que deverá, caso essa mensagem seja desconhecida, proceder ao reenvio de uma mensagem para desmentir a mensagem errada anteriormente enviada. Desta forma pode-se testar o sistema de forma a verificar até que ponto o sistema está preparado para manter a coerência a todos os níveis. A nível de formação esta mensagem serve para treinar os operadores para um cenário de vários operadores ao comando do veículo e onde existe uma falha de comunicação entre os Flight Managers. Isto é um caso que pode criar uma situação de muito risco para o avião e por isso é vital a formação dos operadores.

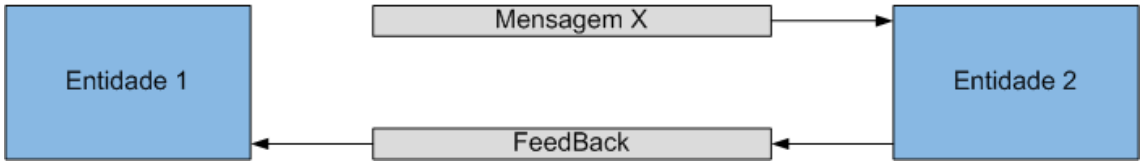

Figura 25: Recepção de feedback de mensagem desconhecida.

3.3.5 Falha por bloqueio de mensagens de feedback

Esta falha é comum e frequente e dessa forma é necessário que o sistema esteja bem preparado para todo o tipo de falhas de comunicação. É normal que numa comunicação exista perda de certas mensagens e por isso é necessário que algumas mensagens exijam feedback. Se estas mensagens não chegarem é esperado que a entidade 1 reenvie a mesma mensagem até receber feedback que essa mensagem foi bem recebida. Esta falha é também importante para a formação de operadores pois é uma falha comum e para o qual é necessário o elemento humano estar treinado para as mesmas situações.

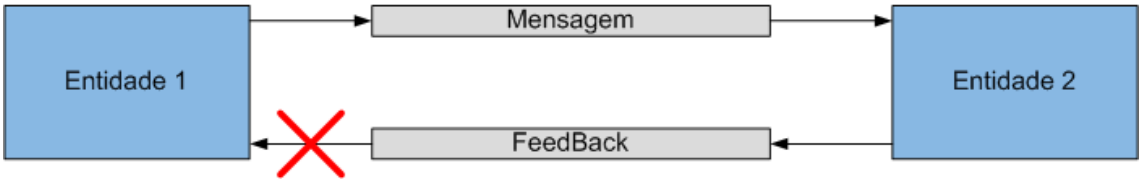

Figura 26: Bloqueio de mensagens de feedback.

3.3.6 Falha por perda de pacotes

Esta é uma falha bastante importante de testar, por exemplo para tramas de informação muito grandes em que o sistema tem de enviar várias mensagens encadeadas. O sistema tem de estar preparado para que as mensagens mesmo que sejam perdidas possam ser recuperadas e depois processadas. Isto acontece por exemplo no sistema de vídeo em que as imagens são enviadas ordenadamente e existem pacotes que se podem perder pela transmissão. Assim sendo é necessário testar e prevenir problemas por causa desta falha. A nível de formação é possível treinar os operadores a adquirirem as melhores configurações de comunicação e a treinar o olho humano a detectar problemas onde por vezes as imagens podem não ser muito nítidas ou mesmo incompletas.

Figura 27: Perda de pacotes.

3.3.7 Falha por recepção desordenada de pacotes

Esta é, tal como a anterior, uma falha bastante comum, e tem uma forma de tratamento bastante simples, ou a ordem é dispensável e no final o sistema reordena as mensagens ou então no caso de transmissão de vídeo a mensagem é descartada e a mensagem apresentada é sempre a mais recente, o exemplo é uma transmissão streaming. A nível de formação de operadores a única vantagem que existe é a habituação a factores de atraso da recepção ou envio de mensagens, perda de informação e de vídeo em qualidade mais fraca e com uma quantidade elevada do chamado flickr.

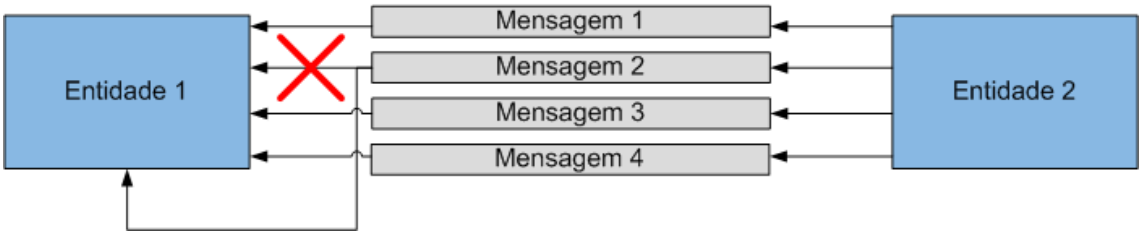

Figura 28: Recepção desordenada de pacotes.

3.3.8 Falha por introdução de erros nas mensagens

Esta é uma falha que pode criar mais problemas. Visto que, de uma forma aleatória, é suposto ao sistema gerar mensagens com erros através da mistura, introdução de novos valores e até modificação da mensagem de modo a que estas fiquem completamente modificadas. Assim o sistema tem de ser capaz de detectar as falhas e de saber que essas falhas ocorreram e voltar a pedir a mensagem à entidade 1. No que toca à formação de operadores, tal como nos anteriores esta serve apenas para treino em novas situações, pois, devido ao carácter aleatório das modificações, o sistema pode gerar estados completamente imprevisíveis e desta forma melhor prepara-los para erros reais.

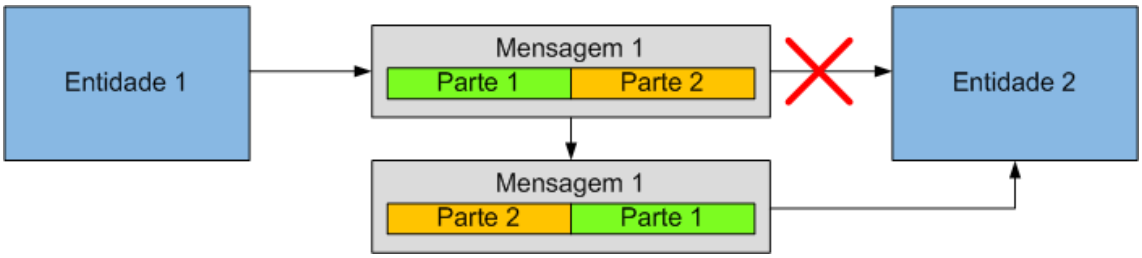

Figura 29: Introdução de erros nas mensagens.

3.4 Resumo e Conclusões

Em resumo serão realizados testes de caixa negra, como por exemplo os testes de integração e de sistemas, bem como teste não funcionais como testes de performance e capacidade, de estabilidade e destrutivos. Todos estes testes têm a mesma utilidade para validar o software e também formar o operador.

São ainda necessárias verificações de falhas como por exemplo bloqueio de mensagens do sistema 1 para o 2, falhas de comunicação, injecção de novas mensagens, falta de recepção de feedback, recepção de feedback de mensagem nunca enviada, falta de pacotes, recepção de pacotes fora de ordem e por fim mensagens adulteradas.

O teste a todas estas falhas de mensagens é um dos objectivos para esta dissertação.

4 Implementação

Neste capítulo será apresentado um conjunto de subcapítulos relacionados com a implementação do sistema, por isso inicialmente será apresentado um subcapítulo onde se fala de como solucionar o problema e seguidamente dos requisitos para este sistema. Nos requisitos falar-se-á de requisitos do utilizador e de mais alto nível.

Finalmente, o capítulo encerra com a apresentação de algumas imagens do programa a correr. Estas imagens são meros screen shots do programa a executar.

4.1 Concepção

Neste capítulo é tratada a forma como a solução foi implementada bem como as hipóteses que foram sendo descartadas ao longo do tempo. De notar que o principal objectivo da solução é a formação de operadores humanos e a validação do sistema para que o nível de confiança no sistema de voo aumente.

Visto então o objectivo de criar uma plataforma dinâmica de teste e validação de software, mas nunca descartando a possibilidade de formar operadores com esta tecnologia começou-se por pensar numa forma de cumprir estes requisitos. Uma vez que o sistema teria de ser testado para que fosse possível encontrar falhas críticas e ao mesmo tempo formar operadores, não era viável fazerem-se estes testes numa missão real, devido ao risco elevado de perdas avultadas de dinheiro e tempo, exemplo disso seria a perda de um aparelho de voo.

Por este motivo tornou-se obvio a necessidade da realização de testes por simulação, pois desta forma não existia o risco de perdas monetárias e ainda de perda de tempo na realização de missões, uma vez que uma missão de teste de equipamentos exige muita preparação além de uma deslocação para uma base aérea segura para evitar danos a civis.

Através da realização de simulações existem duas abordagens possíveis, apesar de só uma ser de possível implementação. De qualquer das formas serão seguidamente explicadas as duas abordagens bem como o porque de uma delas ter sido escolhida e a outra descartada.

4.1.1 Criação de simulador

A primeira abordagem seria através da criação de uma plataforma de simulação de voo, que interagisse com o Piccolo directamente. Desta forma ele pensaria que estaria em pleno voo e por isso colocaria todo o sistema restante numa situação de voo real.

Actualmente já existe um simulador que a CloudCap fornece com todos os Piccolos para que quem esteja a fazer desenvolvimentos sobre este sistema possa testar os sistemas. Actualmente o sistema pode ser representado pelo seguinte esquema.

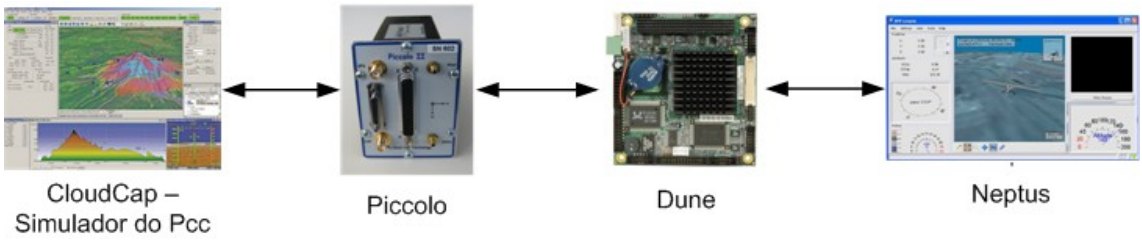

Figura 30: Sistema de simulação actual.

Sendo assim o sistema actual de simulação serve apenas para teste normal de uma missão em condições ideais uma vez que o sistema da CloudCap não possibilita a manipulação de variáveis de voo, teste através da injecção de falhas e teste através da colocação de novos desafios ao sistema. E uma vez que não possibilita nenhuma das opções anteriores os utilizadores formados através deste sistema não saíam completamente preparados para qualquer situação de emergência uma vez que eram treinados para situações normais de operação. Como já referido um dos actuais problemas dos operadores é que estes não estão preparados para assumir o controlo quando algo corre de maneira errada, e sendo assim um sistema destes não formaria operadores capazes de responder a novos desafios. Pelos mesmos motivos o sistema era apenas testado para situações normais de voo, desta forma não poderíamos cobrir um conjunto muito mais abrangente de situações.

Desta forma o sistema actual não é suficiente capaz para o teste do sistema nem para a formação de futuros operadores.

Assim sendo foi pensada numa alternativa, esta seria capaz de gerar falhas e novas situações para que o sistema fosse colocado em situações extremas de voo e por conseguinte os utilizadores também.

A ideia seria criar um simulador de raiz visto que o actual não satisfazia as necessidades reais do laboratório. Este simulador, tal como o actual, funcionaria da mesma maneira mas teria um novo mecanismo incorporado isto é, o sistema seria capaz de ser controlado por um ser humano, desta forma poder-se-ia controlar todo o sistema de simulação e então criar um conjunto de testes que fossem colocar o sistema em situações realmente diferentes.

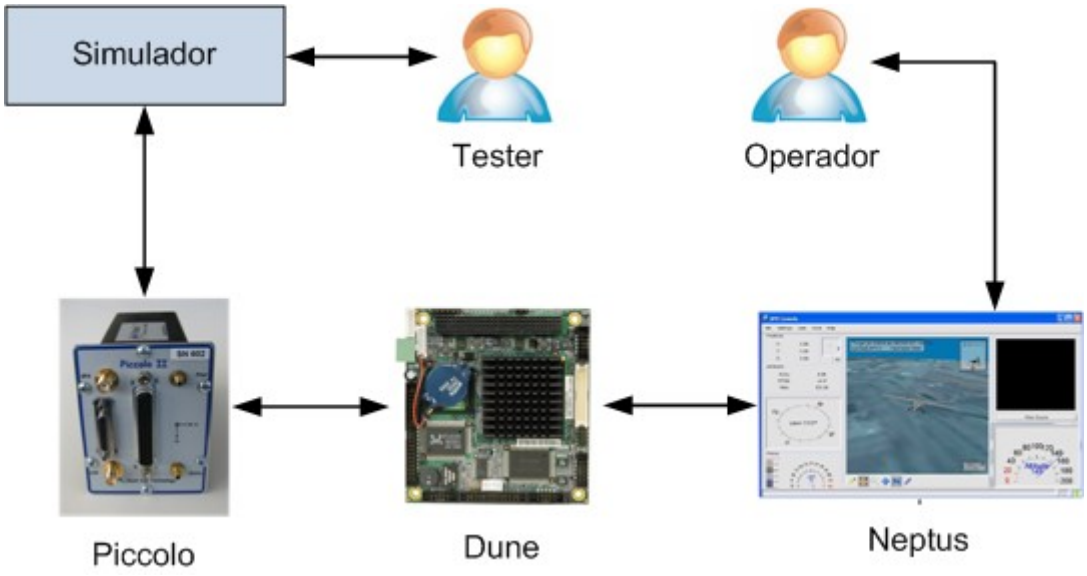

Figura 31: Sistema de simulação pensado para substituir o sistema actual da empresa CloudCap.

Este foi uma solução que foi posta de parte devido ao facto de não ser conhecida a forma de comunicação entre o programa de simulação da CloudCap e o Piccolo uma vez que não existe qualquer tipo de documentação nesse sentido. O simulador actual também não oferece nenhum tipo de possibilidade de introdução de Plugins nem nenhuma forma de modificação possível.

Assim sendo a única forma de descobrir estas comunicações seria através da escuta do canal de comunicação entre estas duas entidades o que seria extremamente custoso e penoso. Desta forma esta solução mostrou-se inviável e por isso foi posta de parte, pois o custo de descoberta das mensagens trocadas iria ser demasiado elevado.

4.1.2 Filtro de Mensagens

Uma vez que não é possível criar um novo simulador ou modificar o existente decidiu-se optar por uma abordagem diferente. Sendo que todos os sistemas são testados individualmente pelos seus programadores não fazia sentido um teste individual de sistemas mas, no entanto uma das falhas existentes no decurso do projecto é a inexistência de uma ferramenta para validação do sistema como um único.

Pegando então nos dois principais objectivos, e em conjunto com responsáveis dos vários sistemas chegou-se a um novo conceito que conjuga da melhor maneira os objectivos globais e individuais dos mesmos. Sendo assim, ao invés da criação de um simulador criar-se-ia um sistema de filtro de mensagens entre os vários sistemas, que permite o teste, validação e também a formação de operadores para situações novas.

Assim sendo, o sistema seria construído como um filtro de mensagens que estaria colocado entre os vários sistemas. Este seria implementado como uma nova tarefa no Dune pois este é o sistema central no projecto e da completa responsabilidade do laboratório e por esse facto de muito mais fácil implementação.

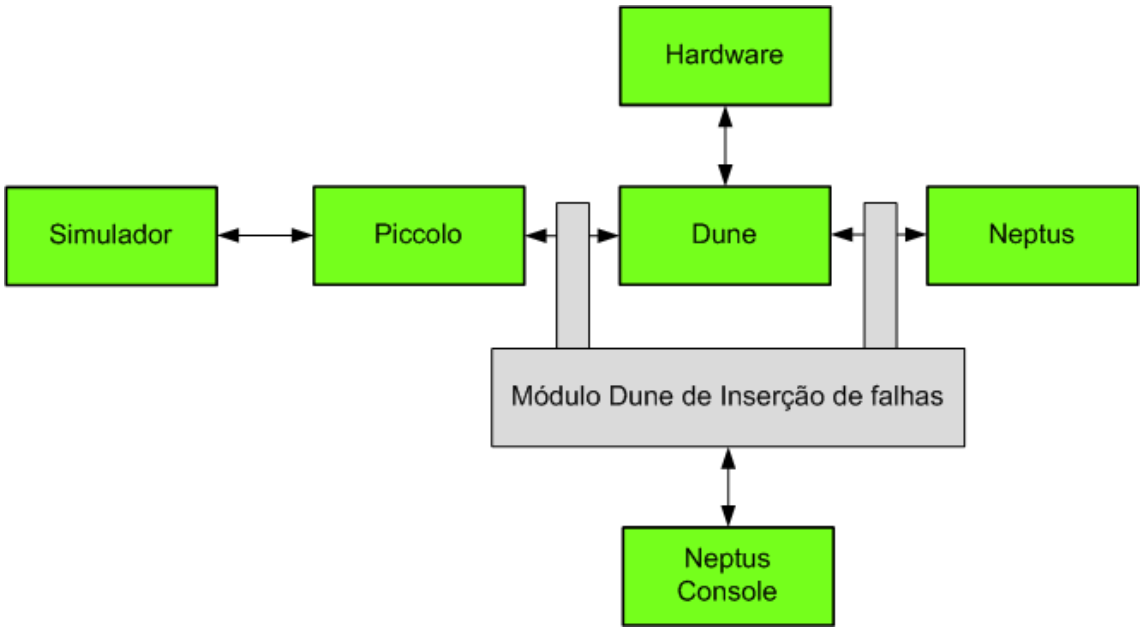

Figura 32: Diagrama do sistema de inserção de falhas.

Como se pode observar o sistema seria colocado no meio dos dois principais canais de comunicação, entre o Piccolo e Dune, o que permitiria bloquear e modificar comunicações entre ambos os sistemas, e ainda entre o Neptus e o Dune, este que tem maior interesse porque para além do teste de software possibilita muitas formas de criar desafios a operadores.

A tarefa seria criada de uma maneira a que todas as mensagens recebidas no Dune, antes de serem tratadas, possam ser modificadas ou não. Desta forma qualquer interacção entre sistemas pode ser bloqueada, atrasada e mudada pelo sistema antes de ser recebida realmente pelo Dune. Desta forma pode-se validar sistemas e ensinar aos operadores como lidar com situações que podem ocorrer ao longo das missões. O sistema teria o seguinte modelo.

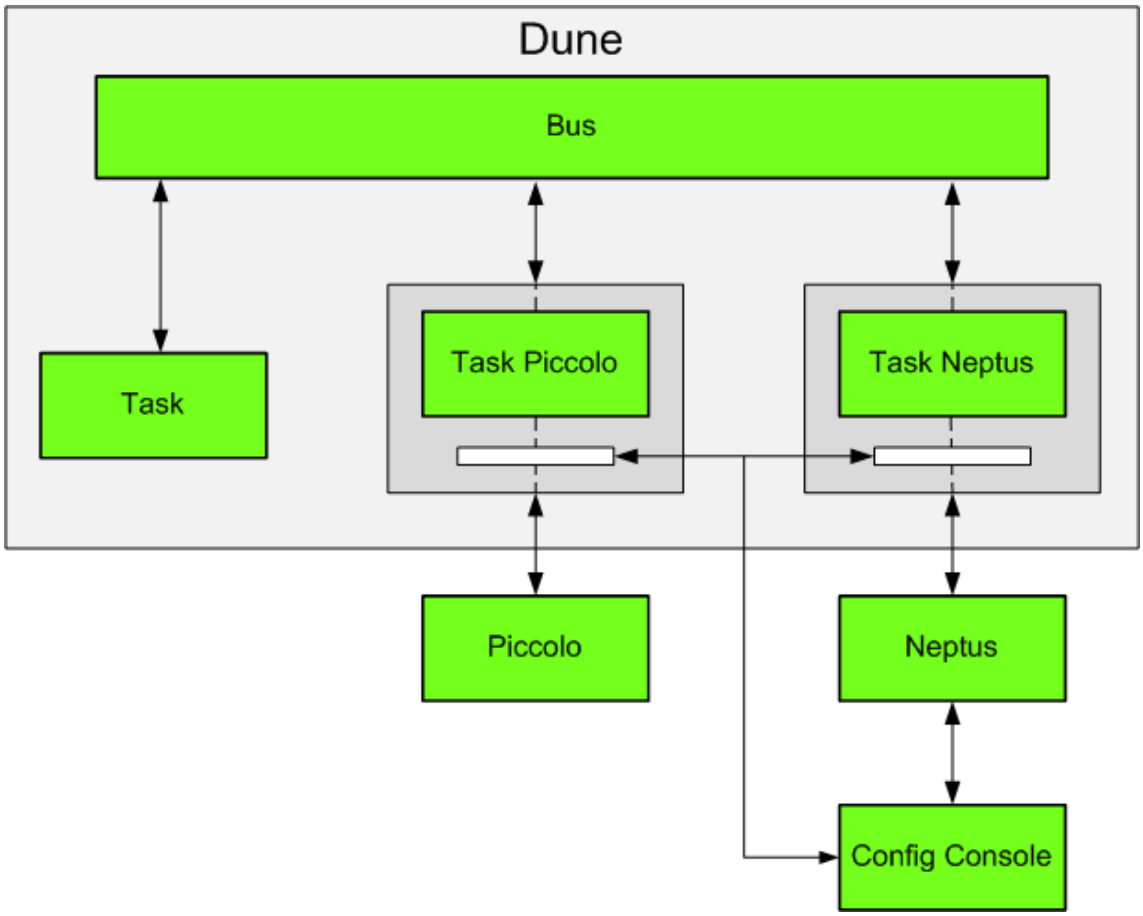

Figura 33: Diagrama de interacção com o Dune.

A nível de interacção o sistema será implementado de maneira a que as mensagens que sejam recebidas pelo Dune passem directamente pelo sistema de falhas antes de serem encaminhadas para o normal decurso das mesmas. Assim o sistema de falhas detém total capacidade de a qualquer momento gerar novas mensagens, bloquear outras e modificar se necessário. A nível de cada tarefa do Dune o sistema funcionará para que todas as mensagens sejam processadas antes de passar para o programa realmente dito.

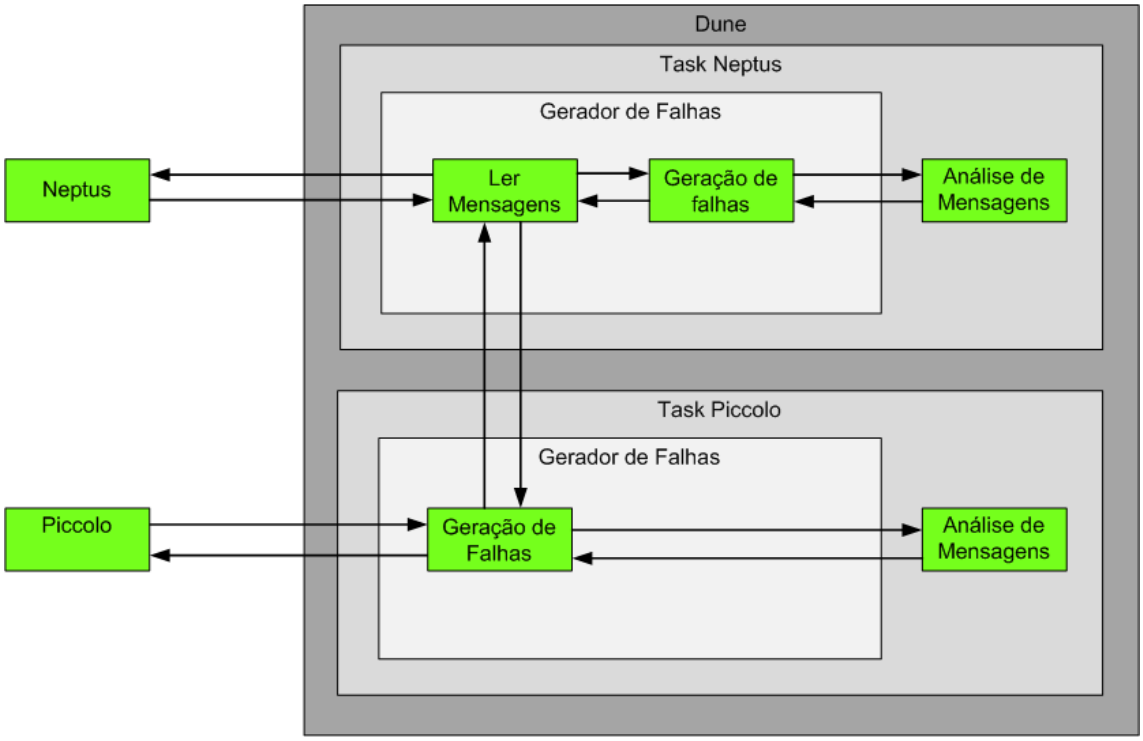

Figura 34: Funcionamento das tarefas no Neptus.

Assim sendo o programa acaba por ser mais um gestor de mensagens que tem a função de criar e gerar novas falhas para que o sistema seja testado. Este tem que ter a capacidade de analisar as respectivas mensagens e falhas de forma a classificá-las como reais erros, ou não.

A nível de interacção com os utilizadores podemos ver como o funcionamento deste sistema como uma caixa negra. Visto que o sistema funcionará apenas como mais uma tarefa do Dune que possuirá a nova capacidade de quer em simulação quer em treino ter uma pessoa a tratar de injectar falhas e a tentar criar erros no sistema.

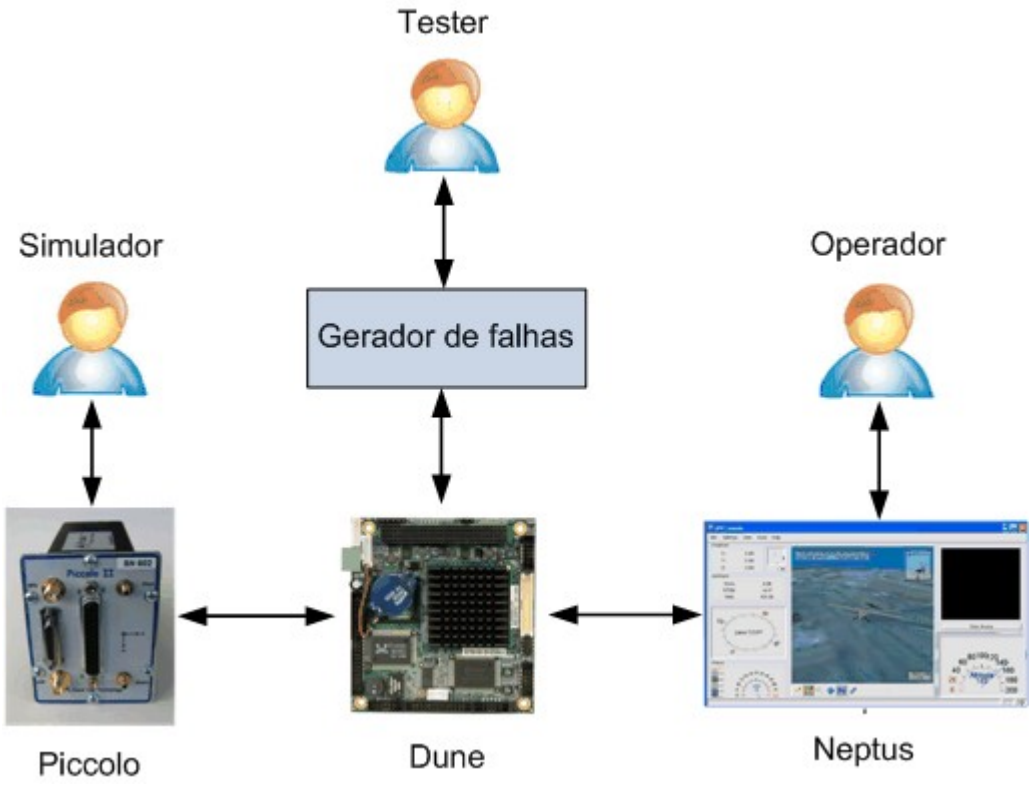

Figura 35: Diagrama de interacção entre os utilizadores o e o sistema.

4.2 Requisitos do sistema

O projecto possui vários requisitos de sistema quer do ponto de vista de utilizador, mas também a nível de funcionalidades pretendidas para que este seja o mais abrangente possível, sendo assim neste capítulo será descrito o que é pretendido para o projecto e para o resultado final do sistema a nível de funcionalidades e requisitos.

Assim sendo o capítulo estará dividido em duas áreas principais, uma de requisitos de utilizador e outra de requisitos do sistema. Os requisitos de utilizador são extremamente importantes pois este é um projecto muito focado no utilizar, pois um dos objectivos principais é a formação de operadores e para teste de plataformas. Por isso é necessário que o sistema seja fácil o suficiente para que o tester consiga fazer debug simples de todos as falhas geradas. No entanto a utilização do sistema para validação do sistema de voo é também vital e por isso a necessária componente de requisitos do sistema.

Para uma melhor compreensão foram usados diagramas de casos de utilização, diagramas de sequência de acções e desenhos de interfaces. Desta forma pretende-se um mais fiel e correcto levantamento de requisitos.

4.2.1 Requisitos do Utilizador

Neste capítulo será apresentado um modelo de casos de utilização pretendido para o projecto bem como a respectiva descrição de cada acção e também será acompanhado de diagramas de sequência para uma melhor percepção dos requisitos a nível do utilizador.

Como já referido o projecto tem como principal objectivo o teste de sistemas e a formação de novos operadores na plataforma Neptus, assim sendo o sistema estará preparado para que um tester consiga interagir com o sistema de forma a poder criar falhas para testar o sistema, ou para o operador reagir e responder correctamente a situações de diversos níveis de risco. Tudo isto sendo executado através de uma simulação de ambiente de voo.

Assim sendo o sistema possui quatro actores, o Flight Manager (director de voo), o operador do avião, o Tester e outros possíveis intervenientes. O Flight Manager está responsável por comandar e gerir toda a missão em que o aparelho se encontra, este é responsável por todas as ordens e todas as decisões do operador devem passar por ele. Este possui uma consola Neptus de comando, onde não pode interagir com o avião mas onde toda a informação relevante é apresentada. O operador está responsável por interagir directamente com o Neptus de forma a pilotar e comandar o avião em segurança num voo simulado, este recebe ordens do director e executa-as. Estes dois elementos são os principais intervenientes para o sistema e como referido o elo deste sistema que tem que receber treino.

O tester possui duas funções importantes, uma é a criação de falhas quando o avião está a ser controlado autonomamente, este necessita de que o sistema esteja em piloto automático e serve para teste de sistemas como o próprio piloto autónomo, e ainda fica responsável por criar falhas quando o flight manager e o operador estão a receber formação, esta formação pode ser específica.

Mais uma vez não foi tido em conta o Piloto porque este set up de voo é o que se espera que futuramente se torne uma realidade, onde o operador toma conta da pilotagem sempre que necessário.

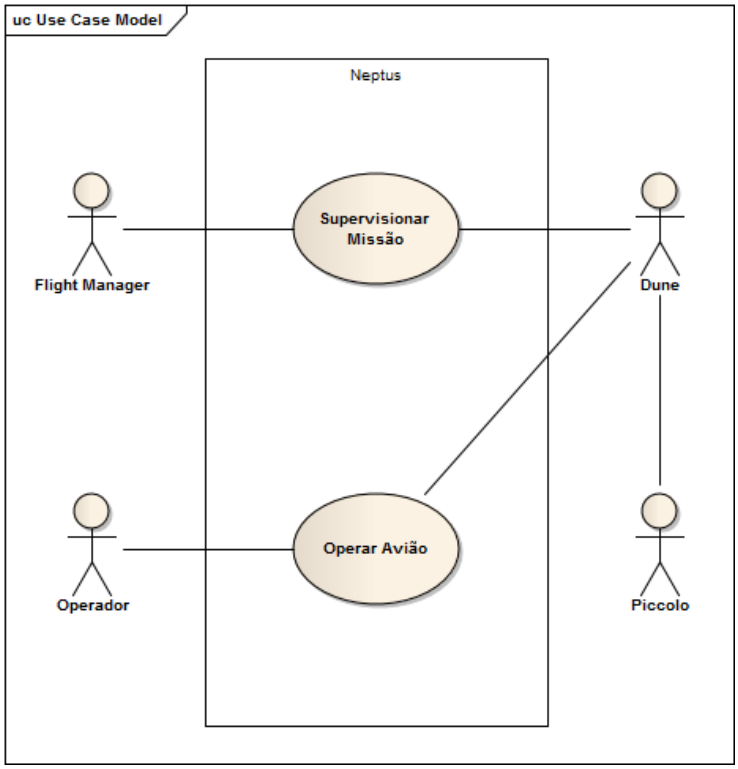

Figura 36: Diagrama de casos de utilização dos utilizadores Operador e Flight Manager.

Sendo assim o Flight Manager tem como objectivo interagir com o Neptus e desta forma supervisionar a missão e comandar as operações. Para o sistema de falhas o actor não passa de mero espectador pois, não interage directamente com o sistema mas, no entanto, é directamente influenciado por ele pois este terá de reagir a falhas e problemas criados por este. A interacção do sistema com este actor é feita através da plataforma Dune e por isso completamente camuflada para este utilizador.

Para o operador de voo este tipo de testes não passam de um treino de simulação de voo normal, com a novidade de que tem de lidar com situações extremas de falhas de comunicação entre os sistemas de voo. Assim sendo o operador será treinado a partir do sistema de voo normal sendo o gerador de falhas, tal como para o director de voo, completamente invisível.

Desta forma ambos não sabem que tipos de falhas estão a ocorrer no sistema bem como que tipos de erros possam ser gerados durante o voo. Assim pretende-se treinar ambos de forma a prepara-los para situações limite, para situações de erro de sistemas e para situações de perigo de perda do aparelho. Isto serve para diminuir o risco de falha humana durante as operações fortalecendo assim o elo mais fraco deste sistema.

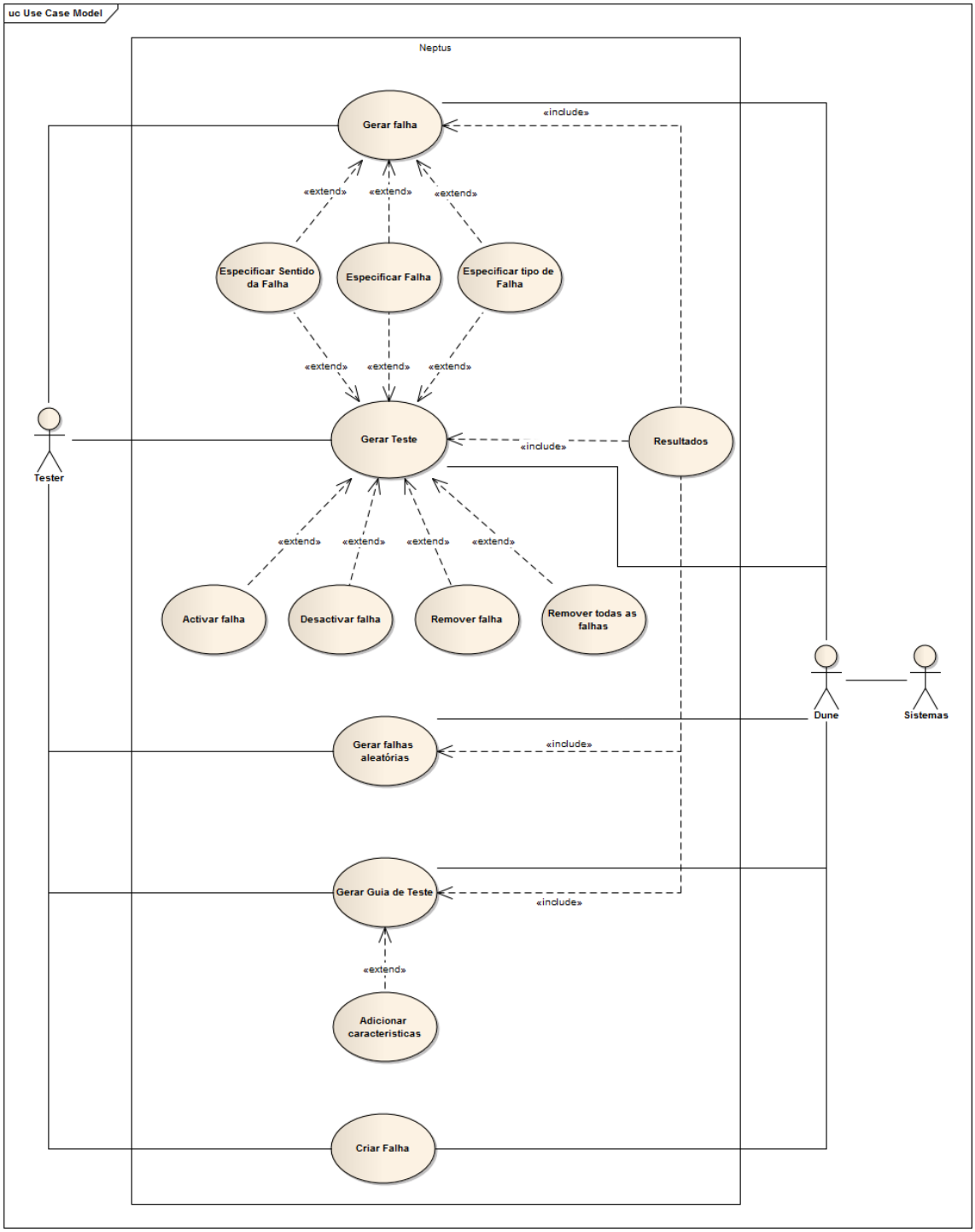

Figura 37: Diagrama de casos de utilização do actor tester.

Para o tester o sistema fica mais complexo sendo este o principal motor de tudo o que se passa de novo no sistema. Assim o tester tem a possibilidade de criar vários cenários de teste, por exemplo pode criar um teste de uma hora com uma taxa de erros de 50 % em que o programa irá gerar erros aleatoriamente em 50% das interacções entre sistemas e verificar que o sistema responde correctamente ao solicitado.

Visto isto o tester pode realizar quatro tipos de operações, gerar falha, gerar teste, criar guia de teste, criar nova. Dentro destes pode realizar diferentes tipos de acções e por isso seguidamente serão apresentados com maior pormenor as operações.

Gerar Falha

Começando por gerar uma Falha, o teste logo à partida tem duas possibilidades, a geração de falha aleatória e a geração de uma falha por si especificada. Se a falha for aleatória o tester simplesmente precisa de indicar essa opção e clicar em gerar que essa falha será gerada e depois mostrado o resultado da mesma.

Se for pretendida uma falha específica o utilizador necessita de seleccionar o sentido da falha e depois disso aparecerá um conjunto de falhas desse género podendo assim o utilizador escolher. Necessita também de indicar o tipo de falha que pretende gerar. Depois bastará clicar em gerar que a falha será gerada e posteriormente apresentado o resultado.

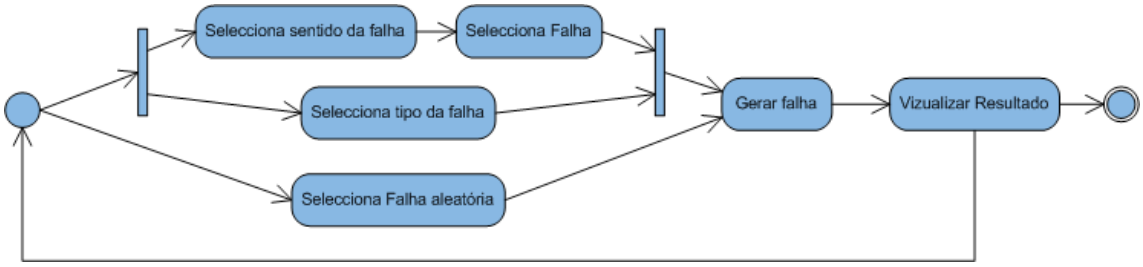

Figura 38: Diagrama de sequência das interacções para o caso de utilização de gerar Falha.

Este tipo de falhas servem para que o tester gere erros em tempo real enquanto os operadores, flight manager e operador, estão em treino de operação no Neptus.

Sendo assim o utilizador neste caso de uso precisa apenas de indicar que quer gerar um erro de qualquer tipo e o sistema gera-o de uma forma completamente abstracta para o utilizador, ou então indicar as respectivas características do mesmo. Seguidamente é apresentado o resultado do mesmo e de uma forma simples para uma mais fácil compreensão por parte do tester.

Gerar Teste

A acção de geração de um Teste suporta dois tipos de interacção com o utilizador, sendo elas a geração de erros aleatórios durante um determinado espaço de tempo e com uma taxa de falhas associada, e a geração de vários testes específicos sendo activados e desactivados pelo

utilizador, este também comporta a necessidade de indicação de uma duração e percentagem associada.

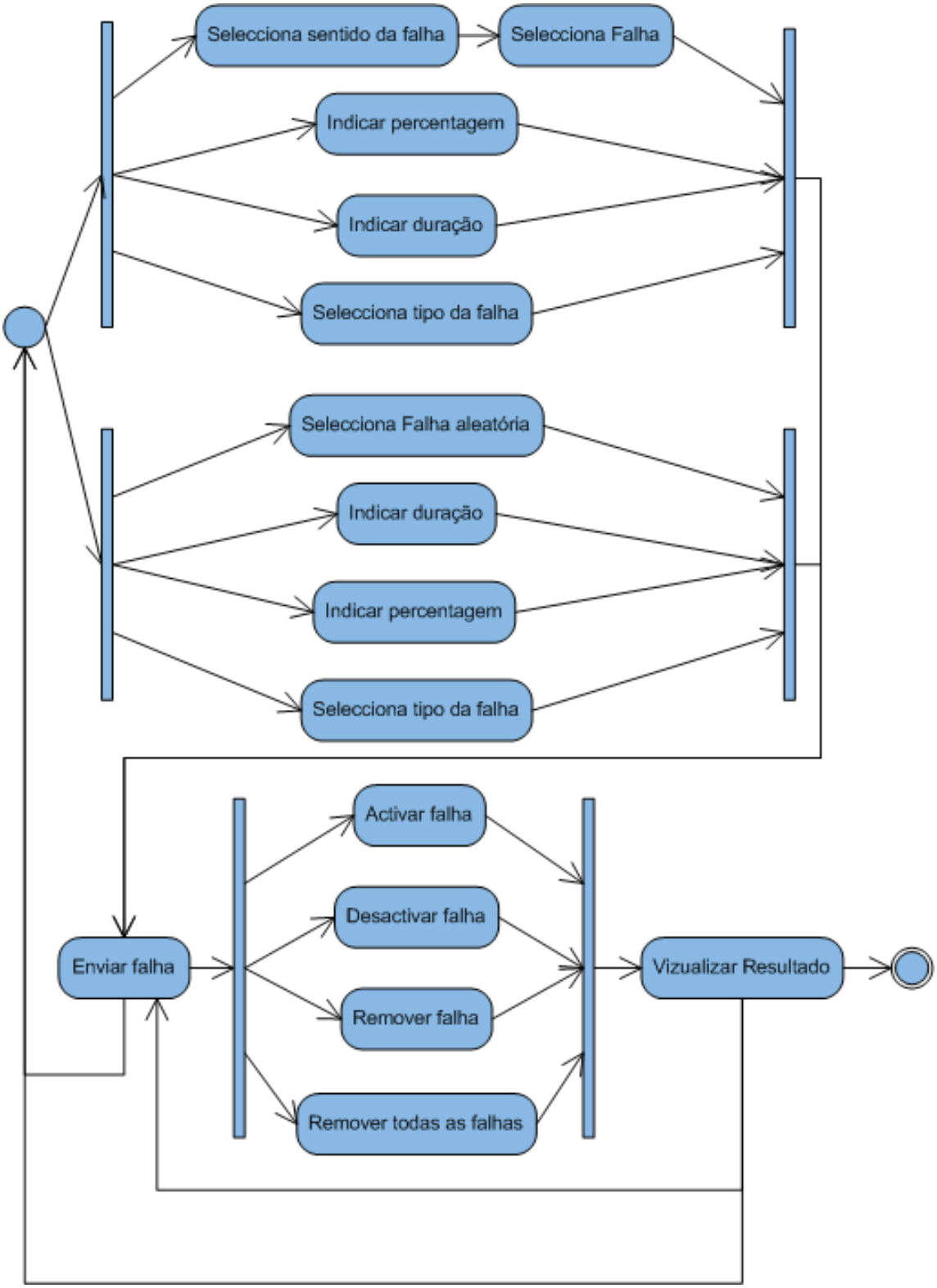

Figura 39: Diagrama de sequência das interacções para o caso de utilização de gerar um teste.

Geração de falhas aleatórias num espaço de tempo definido

A nível da interacção temos então a geração aleatória de erros durante um intervalo de tempo determinado e com uma taxa de erro associada. O teste é executado pelo tester em que este tem apenas de indicar o intervalo de tempo, a percentagem de falhas, tipo de falha e que quer gera-los aleatoriamente, depois o programa durante esse tempo gera uma percentagem de erros igual à indicada e vai mostrando-os na janela de visualização.

Este tipo de interacção servirá quer para fazer stress test ao sistema podendo mesmo gerar um cenário de várias horas de teste intensivo para verificar o comportamento ao longo do tempo. Mas também será possível gerar um stress test não só para o sistema mas também para os utilizadores em treino.

O utilizador nesta acção pode, independentemente da ordem, definir o intervalo de tempo, o tipo de falha, a opção aleatória e a taxa de erros gerados. Depois de todos os campos preenchidos o utilizador aguarda e os erros vão sendo mostrados no ecrã até o teste acabar. À medida que são mostrados os resultados também é conhecido se estes falharam ou passaram.

Geração de falhas específicas num espaço de tempo definido

Finalmente a acção geração específica de falhas durante um intervalo de tempo determinado e com uma taxa de falhas. Este tipo de teste serve para uma exaustão de um certo tipo de teste sobre mensagens. Como no anterior o utilizador só necessita de indicar ao sistema que tipo de falha gerar, a taxa de criação de falhas, o tipo de falha e o espaço de tempo que as pretende gerar, depois o sistema indicará o sucesso um insucesso dos mesmos.

Neste tipo de interacção o utilizador precisa de indicar a falha a gerar, o intervalo de tempo, o tipo de falha e a taxa de falha. Inserção que é independente da ordem. Depois bastará ao utilizador aguardar que o teste termine e que sejam apresentados todos os resultados no ecrã.

Esta operação é acompanhada por cinco botões sendo eles, enviar que serve apenas para enviar para o Dune o objectivo do teste, o activar que serve para iniciar um teste depois de seleccionado qual o que se pretende activar, o desactivar para o objectivo contrário do anterior, o remover para remover a configuração do Dune e finalmente remover todos para remover todas as configurações anteriores no Dune. Estes últimos quatro botões podem ser usados a qualquer momento do teste desde que seja seleccionado uma configuração já no Dune.

Criar guia de Teste

Este é um tipo de teste bastante específico e prende-se com a necessidade de fazer testes encadeados e com a necessidade de fazer vários testes de forma conhecida e explicita para o tester. Funciona como uma espécie de Batch de testes de forma a obter resultados mais precisos.

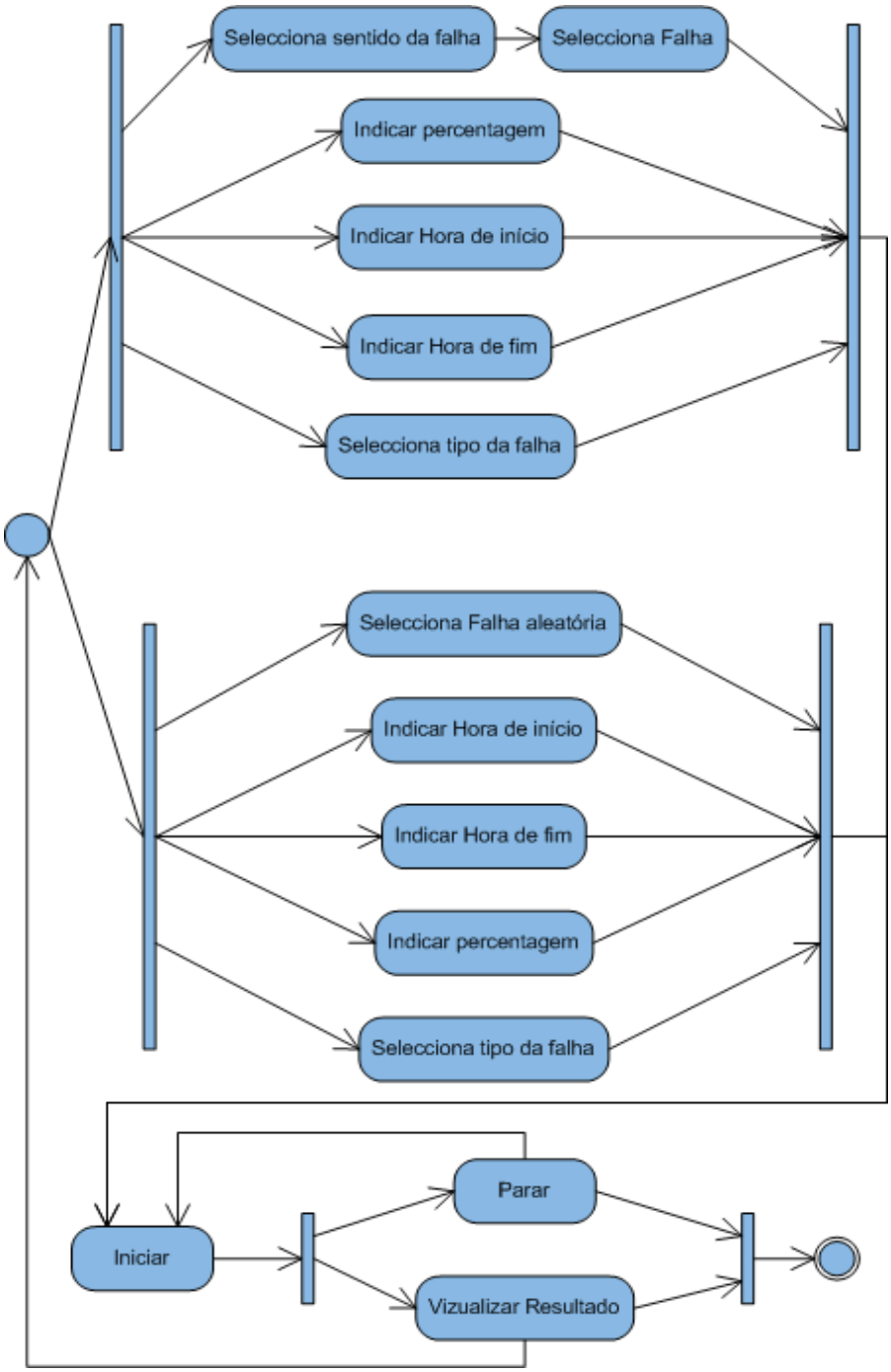

Figura 40: Diagrama de sequência para o caso de utilização de criar um guia de teste.

Sendo assim, tal como o caso de uso anterior o utilizador tem a possibilidade de escolher se quer uma geração aleatória ou especifica de falhas, a única diferença está no numero de vezes que a definição da falha é feita. Através de uma tabela, estes dados são passados com uma hora

de início e uma hora de fim para que o utilizador saiba de forma mais simples como sincronizar as várias falhas. Sendo assim os utilizadores necessitam de dar as mesmas informações que no caso de uso anterior e depois clicar em iniciar, este correrá cada uma das premissas até o final. Este terá sempre atenção a hora e a taxa de falhas para que tudo corra de acordo com o pretendido. No final como sempre mostrará os resultados do teste.

Criar uma nova falha

Finalmente a criação de uma nova falha serve para que seja possível de forma simples e fácil qualquer utilizador adicionar novas mensagens e novos tipos de testes serem adicionados de forma rápida.

Isto porque este ainda é um sistema em desenvolvimento e, por isso, em constante mudança e crescimento. Estão sempre a surgir novas mensagens e novas interacções entre os sistemas e por isso surge a necessidade de novas capacidades de teste do sistema. Assim sendo o utilizador pode adicionar uma falha que testa uma mensagens ou várias mensagens de forma a tornar os testes mais encadeados.

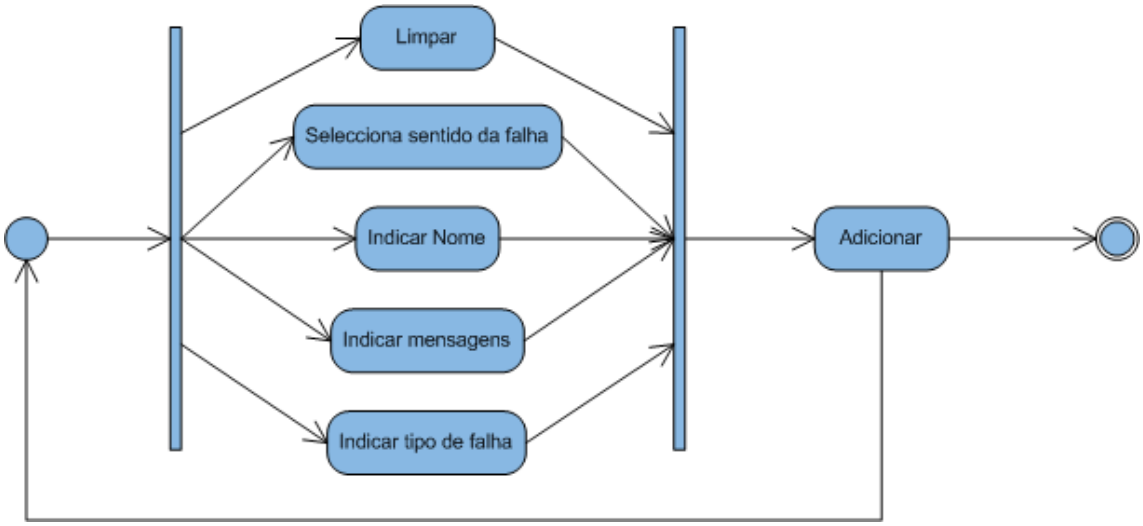

Figura 41: Diagrama de sequência das interacções para o caso de utilização de criar uma nova falha.

Visto isto o utilizador necessita apenas de indicar um nome, uma ou várias mensagens associadas e indicar o tipo de falha pretendido, este é dispensável pois pode ser indicado no decorrer do teste.

4.2.2 Interface com o utilizador

Neste subcapítulo será abordada a parte de interfaces com o utilizador, visto ser um sistema muito virado para o tester e para a formação de operadores é fulcral que a criação e avaliação de falhas seja simples e prática, quer de analisar quer de gerar novas falhas. Assim sendo a geração de falhas será constituída por um conjunto de 3 interfaces assentes numa interface principal.

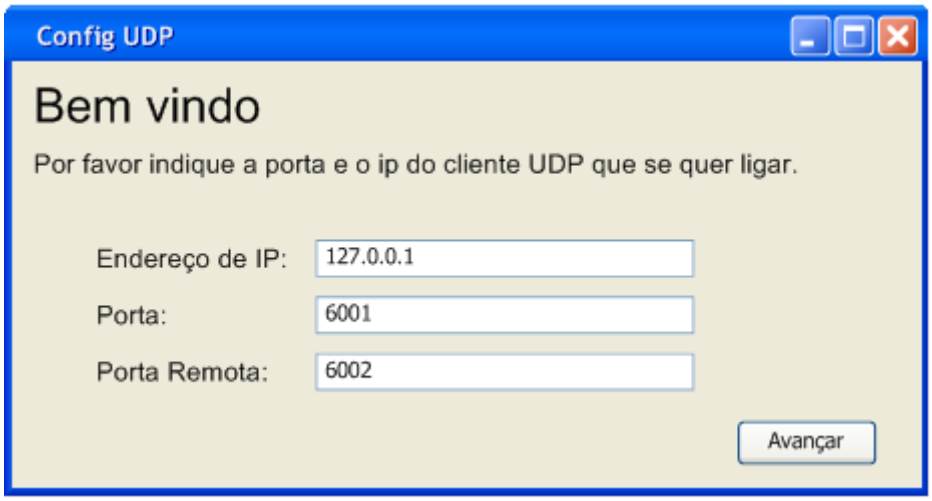

Figura 42: Maquete da interface de início do programa.

Esta será a primeira interface, tem como objectivo a definição dos parâmetros para as comunicações entre o programa e o Dune. Nesta pode ser introduzido o IP da máquina onde o Dune corre bem como a porta e a porta remota para o envio e recepção de mensagens.

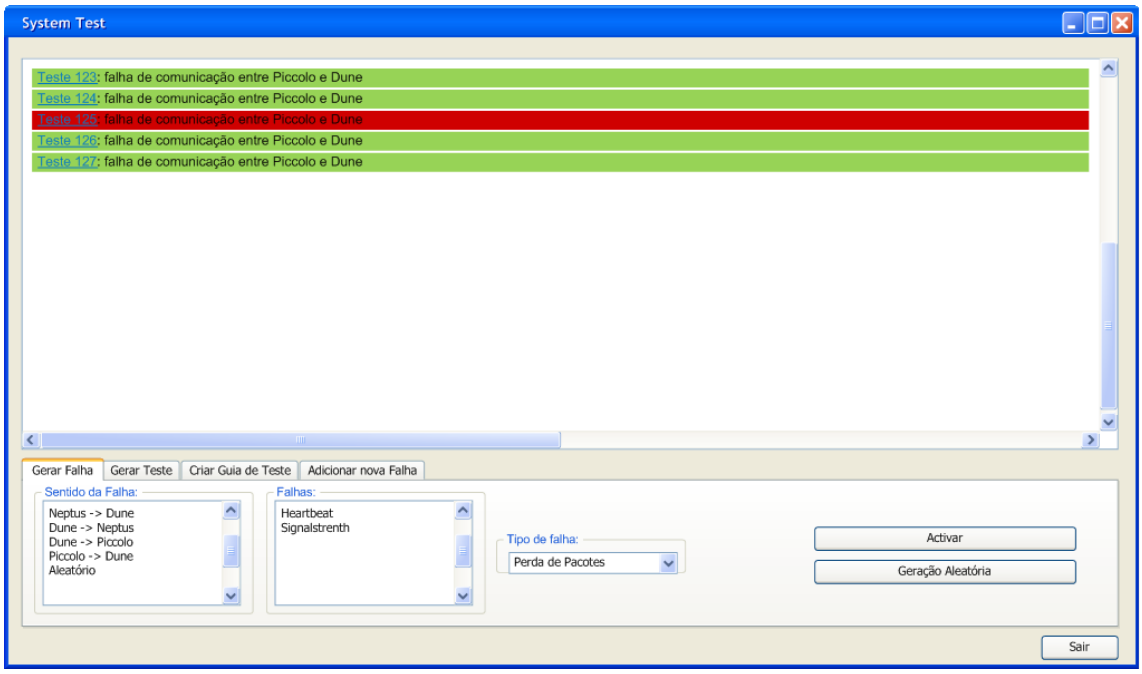

Figura 43: Maquete da interface de gerar uma falha.

Este é a interface para a geração de uma falha, nela podemos indicar o sentido da falha, a falha e o tipo de falha a ser gerado. Existem ainda dois botões, um para activar a falha seleccionada e outro para gerar uma falha completamente aleatória. Assim existe a possibilidade de durante os treinos de pessoal facilmente indicar as falhas pretendidas para cada momento.

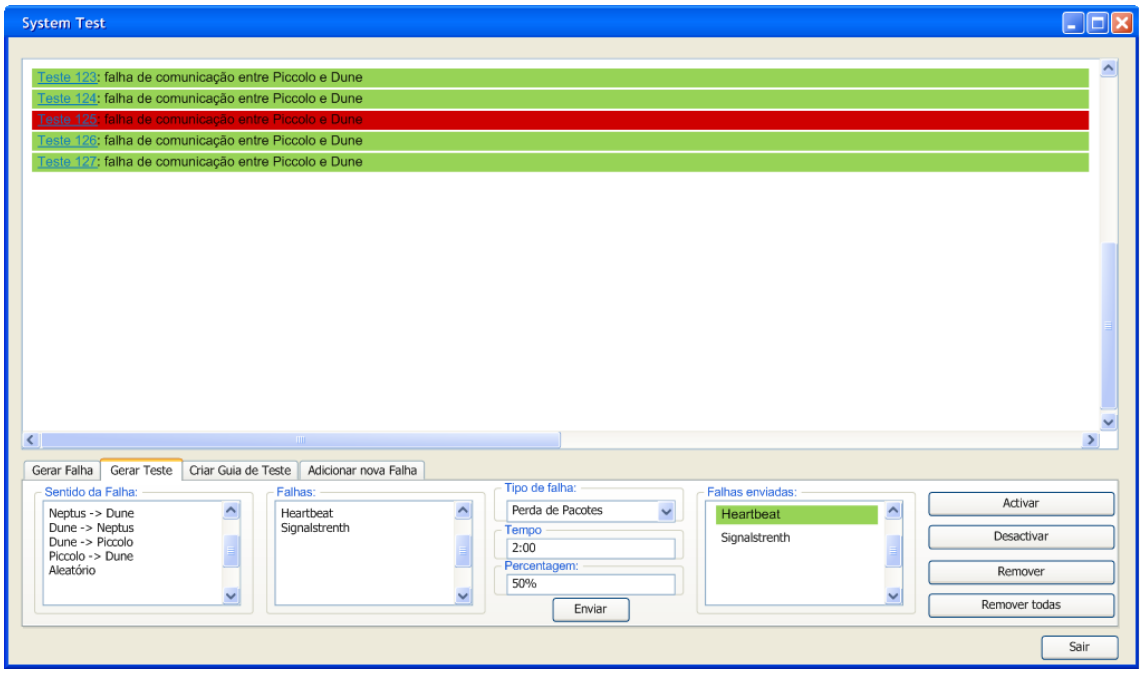

Figura 44: Maquete da interface de gerar um teste.

Esta interface tem como objectivo a geração de testes que tal como já foi referido necessita da indicação primeiro da falha que se pretende configurar, depois o tempo, a percentagem e o tipo de falha. Depois esta informação é enviada e passa a estar disponível para activação. Depois é ainda possível desactivar a falha, remove-la e mesmo remover todas se pretendido. De notar que se for pretendido gerar falhas aleatoriamente basta indica-lo no sentido e depois acrescentar a informação da percentagem e duração. Depois basta enviar e activar essa falha para que o sistema passe a gerar aleatoriamente falhas durante esse tempo e com essa percentagem de falhas.

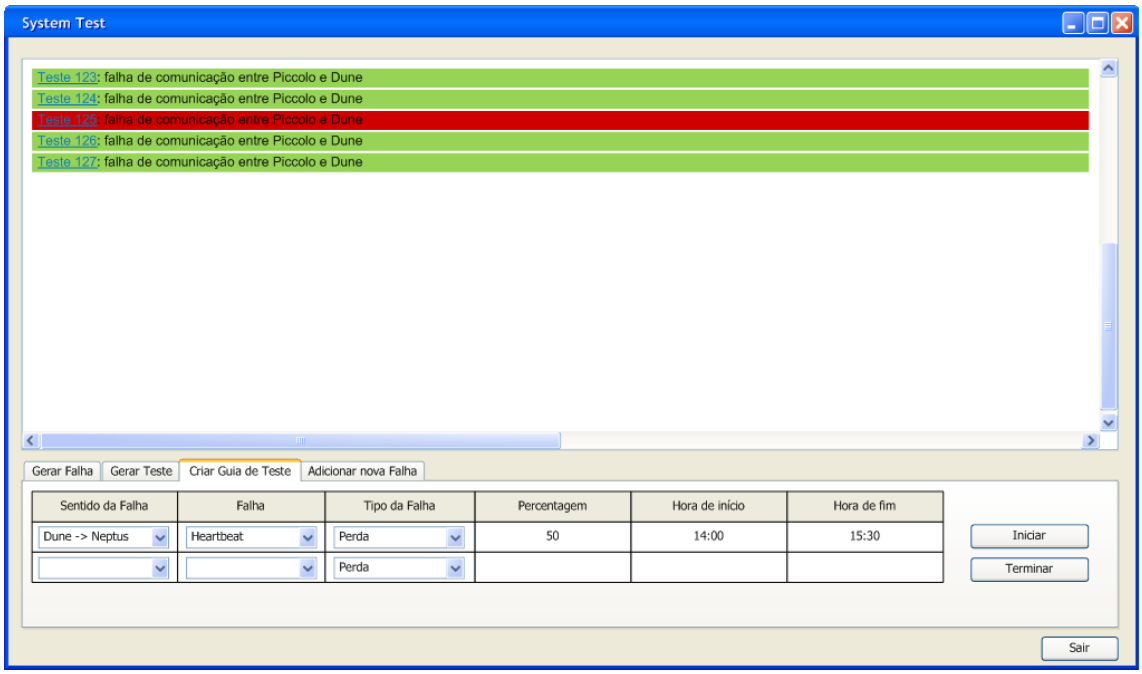

Figura 45: Maquete da interface de criar um guia de teste.

Esta interface tem como objectivo criar um guia de teste e para isso usa uma tabela onde constam combobox para a mais fácil selecção das falhas. Também é possível seleccionar falhas aleatórias tal como na interface anterior, a única diferença é que existe uma hora de início e uma hora de fim, em vez da duração.

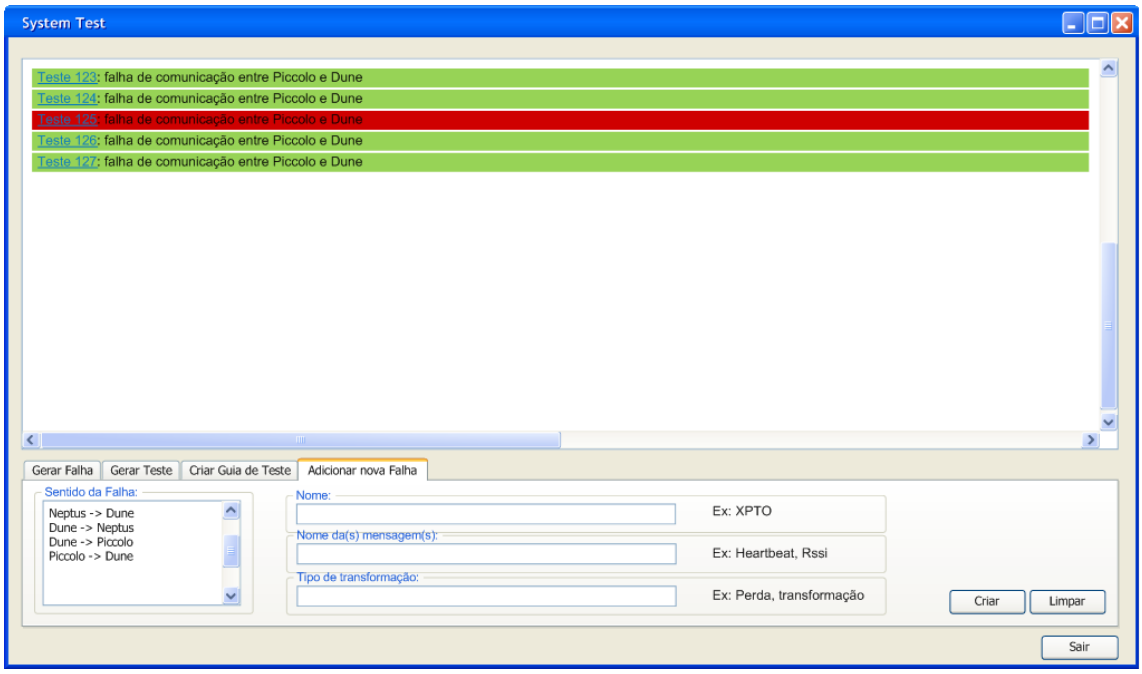

Figura 46: Maquete da interface de criar uma nova falha.

Para a criação de uma nova falha basta só indicar o novo nome, as mensagens contempladas e finalmente o tipo de falha que é dispensável se desejado pelo utilizador. Existem ainda dois botões para finalizar a operação ou então limpar todos os campos de forma mais rápida.

A janela em si contém um botão para sair da aplicação e uma janela de Tabs para mudar entre todas as opções do programa. Bem como a janela de visualização de resultados deve conter cores diferentes para que seja fácil de encontrar falhas que resultaram em erro.

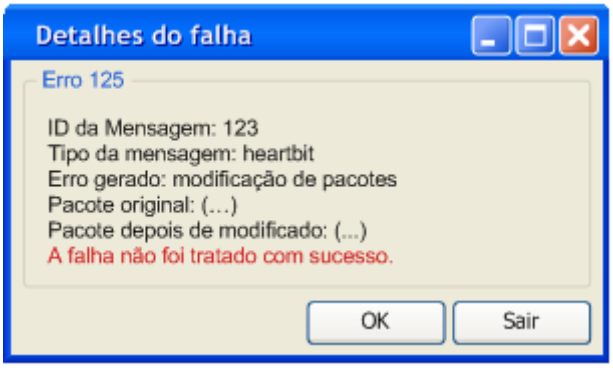

Figura 47: Maquete da interface de detalhes da falha gerada.

Esta é a consola com os detalhes da falha gerada. Nesta consola poderá ser analisada a falha gerada, bem como o que deveria ter acontecido e o que realmente se passou para efeitos de debug do sistema, por exemplo na simulação de perda de pacotes importantes, se o pacote não for reenviado o sistema possui um erro, e então a consola possuirá essa informação.

4.2.3 Requisitos funcionais do Sistema

Como já mencionado o principal objectivo do projecto é a criação de um motor de geração de tipos de falhas de forma a certificar e a testar todos os sistemas. Terá uma forma intensiva de teste e uma forma sistemática e continuada. Isto permitirá certificar e formar utilizadores ao mesmo tempo que se certifica o sistema.

Assim sendo o sistema tem de ser capaz de gerar falhas. Estes deverão ser os mais abrangentes possíveis para que este seja uma boa plataforma de teste e validação do software e hardware de todo o sistema de controlo. Estas falhas terão de ser objectivas e coerentes pois têm como objectivo a detecção de problemas do sistema de controlo e também para treino de operadores em situações limite de falha de controlos de comando e de erros do sistema.

Os erros terão de ser os mais vastos possíveis de modo a possibilitar uma maior probabilidade de detecção de falhas e também formação de operadores para o maior conjunto de situações extremas possíveis.

Este erros para além de gerados têm obrigatoriamente de ser analisados para se perceber qual a consequência da falha gerada, visto que o sistema pode responder positivamente nunca entrando em estado de erro, mas pode também criar um erro e esse tem que ser de fácil interpretação para efeitos de debug. Isto torna-se muito importante para o sistema porque assim o tester só precisa de tomar em atenção as falhas que geraram casos de erro no sistema. Assim, serão de mais fácil correcção pois toda a informação referente à falha é guardada como um registo de eventos.

O sistema terá de gerar sequências de falhas e de maneira contínua. Só assim o sistema poderá de forma continuada e sistemática e abstraída do utilizador analisar as possíveis falhas do sistema. Isto é importante porque as mensagens não são sequenciais e por isso podem haver falhas que o programa ainda não sabe se passaram ou falharam mas, já está a criar uma nova falha. Sendo este um desafio bastante grande por parte do programa conseguir a sincronização de todo o sistema.

O sistema tem de ser capaz de gerar falhas para que seja possível formar operadores pois este é um dos objectivos principais e fundamentais do mesmo. No entanto o teste de sistemas nunca poderá ser descartado, sendo ambos de vital importância para o projecto.

Sendo assim o sistema terá de ser capaz de gerar para realizar testes de stress, testes por introdução de falhas e testes de fiabilidade do sistema. No que toca à formação tem de ser capaz

de usar os seus erros para criar situações de perigo e stress para que o utilizador aprenda a lidar com esses mesmos cenários que podem acontecer na vida real.

4.3 Resultados

Apesar de ainda numa fase não definitiva o programa implementado possui já bastantes opções já a funcionar e por isso já é possível realizar testes. Infelizmente não existiu oportunidade de tal acontecer devido a atrasos na conclusão da ferramenta e também à impossibilidade de montagem de um set up de simulação de voo no laboratório por causa do aproximar de missões e demonstrações do projecto a entidades exteriores.

No entanto neste capítulo vai ser mostrado o que actualmente está a correr bem como a forma de gerar falhas e como interagir. Para esta demonstração foi usado um pequeno conjunto de mensagens só mesmo para ser possível o teste de software.

Sendo assim, e começando por falar da primeira janela visível mal o programa seja arrancado, temos então a janela de configuração para a ligação ao Dune. Uma vez que o sistema usa mensagens do IMC para comunicar com o sistema que corre na PC-104.

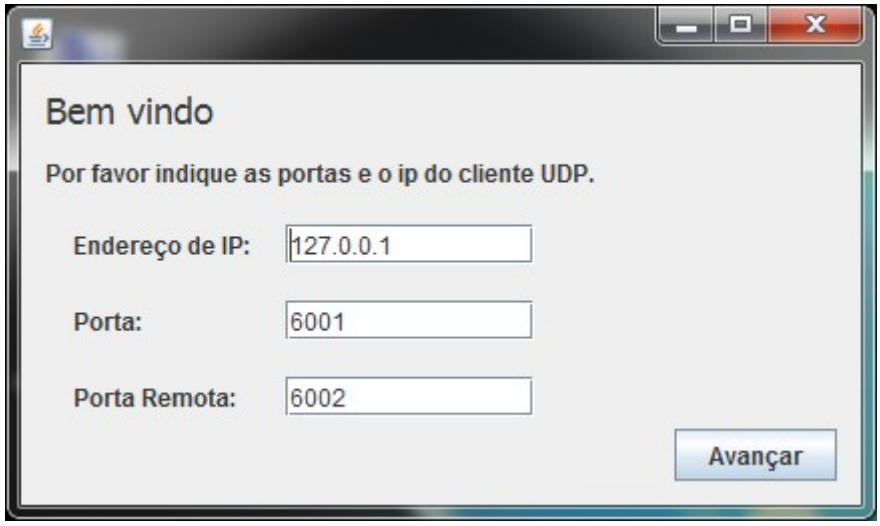

Figura 48: Janela de configuração das comunicações UDP.

Seguidamente, é apresentada a janela onde estão todas as opções de teste. Nestas janelas está tudo preparado para a realização dos testes menos a geração automática quer de testes quer de falhas. Também de realçar que a janela de configuração de guia de testes não está completamente operacional. Sendo assim são demonstradas as janelas do sistema, já com alguns erros gerados. De salientar que para verificar as suas características basta clicar num dos itens.

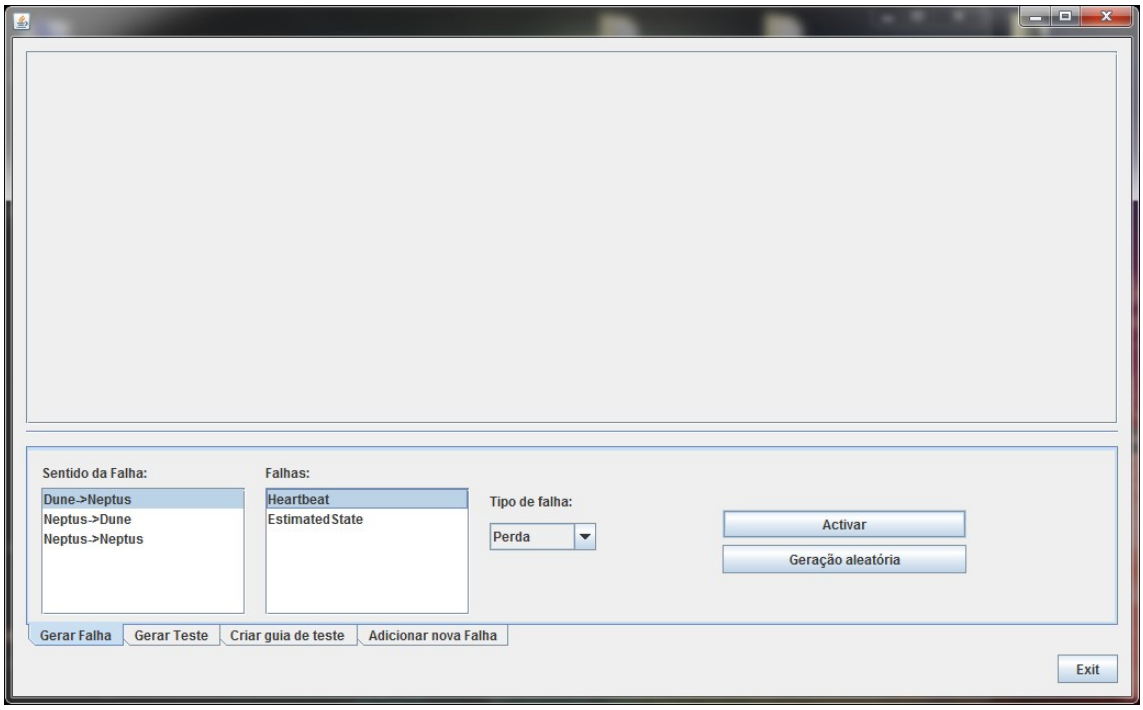

Figura 49: Janela de criação de uma falha.

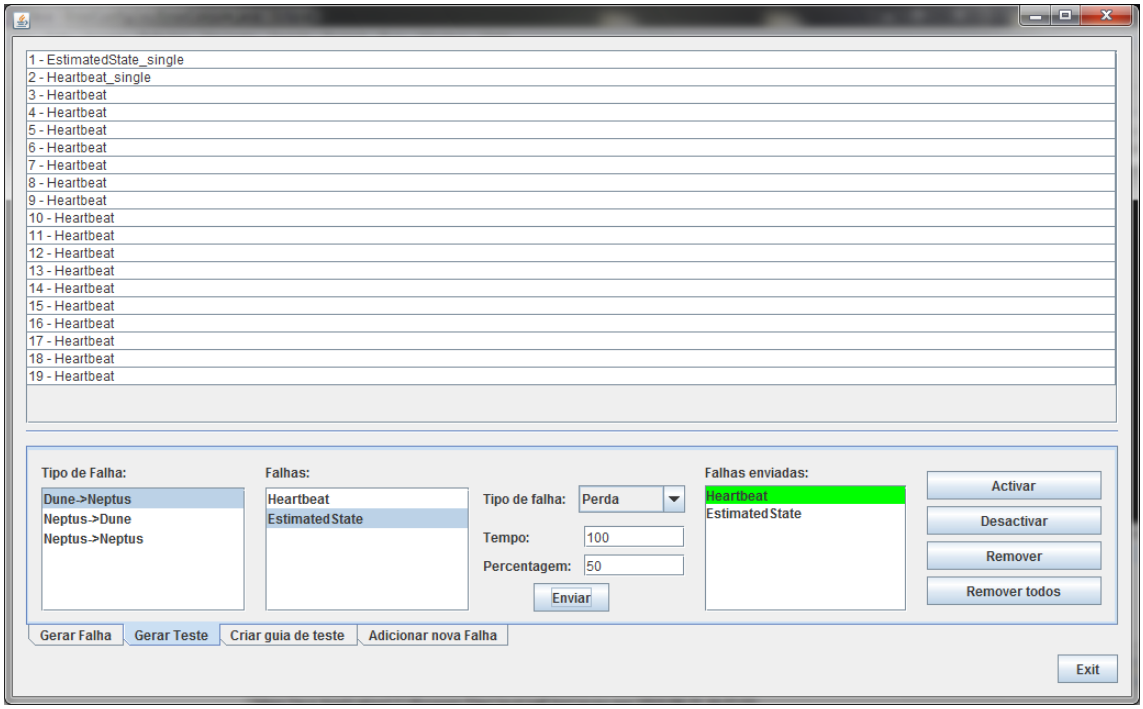

Figura 50: Janela de criação de um teste.

Neste menu, a pequena secção de falhas enviadas é dinâmica, isto é sendo uma falha activada esta fica verde, sendo desactivada fica de cor normal e sendo removida esta desaparece.

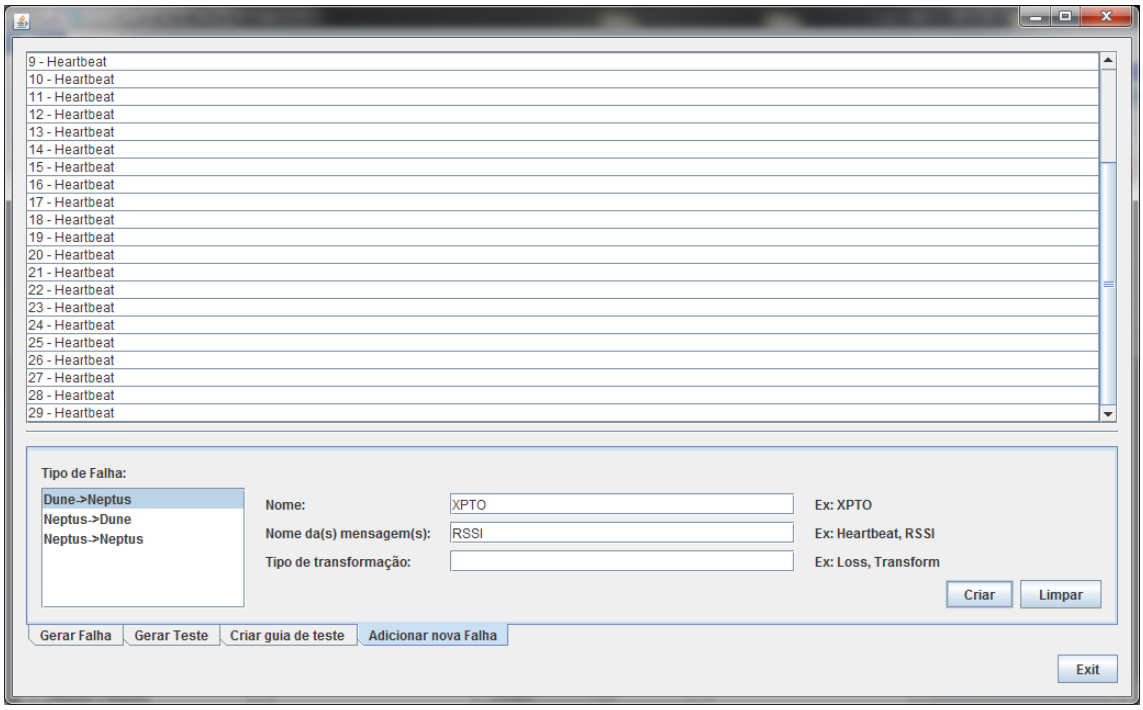

Figura 51: Janela de adição de uma nova falha.

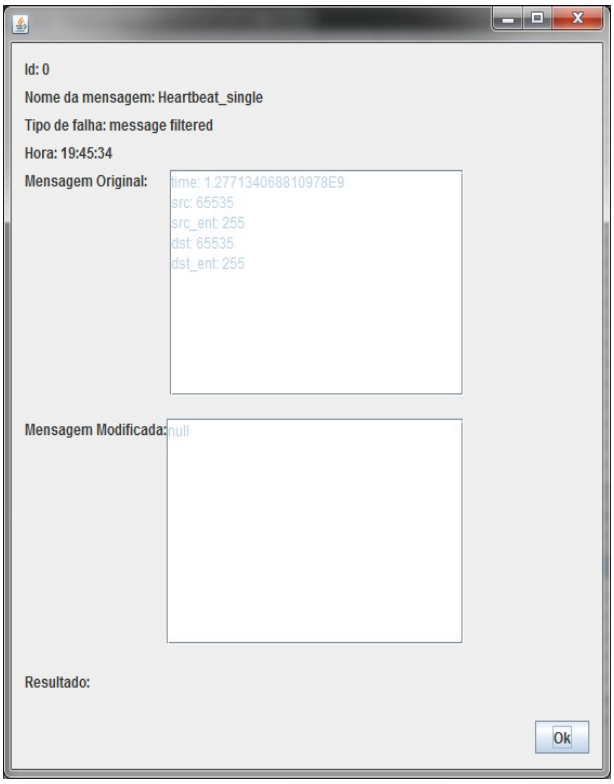

Figura 52: Janela de visualização das informações das falhas.

4.4 Resumo e Conclusões

Entre as duas opções possíveis, ou a criação de um simulador de voo, ou a criação de um filtro de mensagens entre sistemas, optou-se por criar o segundo. Isto deve-se ao elevado custo de tempo que o primeiro iria ter, isto porque como não existe documentação seria impossível, ou quase, o conhecimento de todas as mensagens trocadas ente o sistema de simulação e o Piccolo pela porta CAN.

Assim sendo foi estabelecido um conjunto de casos de uso e de objectivos que foram cumpridos em grande parte. Na conclusão serão apresentados que objectivos foram realmente cumpridos ou não.

5 Conclusões e Trabalho Futuro

Os sistemas críticos estão cada vez mais presentes no nosso dia-a-dia e por isso é cada vez mais necessário garantir toda a sua segurança. Uma vez que cada vez mais estes sistemas estão a interagir diariamente com os utilizadores, existe uma necessidade extra de que estes sistemas nunca falhem, pois como já foi referido quando os erros acontecem os efeitos podem ser devastadores.

Cada vez os sistemas são mais complexo e mais responsabilizados pela tarefa ou tarefas que desempenha e cada vez mais tarefas extremamente complexas lhes são delegadas. Assim sendo, as empresas cada vez mais procuram ferramentas mais fáceis e versáteis de teste, uma vez que a validação de sistemas é extremamente moroso, complexo e dispendioso. Assim sendo as empresas procuram sistemas fáceis, baratos e eficazes o teste de software crítico.

Pode-se concluir que esta é uma área de gigantes. Se pensarmos no sector aeroespacial, empresas como a NASA, ESA, Boing e Airbus dominam um mercado difícil de ser integrado por muitos outros. Os seus sistemas de teste são completamente desconhecidos pois na maior parte dos casos são projectos secretos e construídos para os seus próprios sistemas. Estes sistemas são fruto de anos de investigação e aprimoramento de sistemas que não podem falhar em momento algum.

Figura 53: À esquerda uma plataforma de teste de voo da NASA, ao centro o cockpit de um simulador de teste da Airbus e à esquerda um simulador do Boing.

Estes sistemas são quase perfeitos em teste e formação dos seus operadores, mas no entanto são caros, demasiado complexos e demasiado pesados para o sistema que é pretendido testar neste projecto. Nem estes sistemas estão preparados para correrem em computadores normais como também não estão preparados para interagir com o hardware usado neste projecto.

Assim sendo a criação desta nova plataforma faz todo o sentido. Como foi referido o sistema para além de oferecer as capacidades de teste e validação de sistemas, a plataforma teria também de apresentar uma forte componente de formação de operadores, e por isso foi necessário desenvolver um sistema à medida, pronto para interagir com os sistemas actuais do laboratório bem como pronto para interagir com um possível utilizador num terminal de controlo.

Os objectivos deste projecto foram amplamente cumpridos apesar de algumas funcionalidades ainda não estarem completamente implementadas. No entanto é uma questão de tempo até que todos os objectivos sejam cumpridos e o programa passe a fazer parte do dia-a-dia do laboratório.

5.1 Satisfação dos Objectivos

Como já referido acima os objectivos foram amplamente cumpridos para este projecto, pois para além de já existir uma ferramenta criada que possibilita o teste e a formação de utilizadores, foi executada uma ampla análise de mensagens e de ferramentas para se chegar à melhor solução possível. De notar que este sistema foi realizado à medida para este projecto por isso existiu uma forte componente de estudo envolvida quer no estudo de mensagens quer no estudo de soluções. No entanto existem ainda vários pontos a melhorar e a rectificar, bem como muitos outros a terminar.

Assim sendo os objectivos cumpridos com sucesso são:

- Criação de uma falha única;
- Criação de um teste com uma duração e taxa de erro associada;
- Adição de novas falhas;
- Interacção com o Dune em runtime;
- Comunicação entre a plataforma fixa de teste e o Dune através de mensagens IMC;
- Tratamento de mensagens no Dune;
- Visualização de resultados em tempo real;
- Possibilidade de criar mais falhas, mesmo já antes correndo um teste;
- Capacidade de autonomia em teste dinâmico de várias horas;
- Capacidade de visualizar as mensagens antes e depois de serem tratadas.

No entanto existem alguns objectivos que não foram totalmente cumpridos.
Sendo as principais:

- Avaliação da falha, para saber se esta gerou um erro ou não;
- Geração de vários tipos de falhas para alem de perda de pacotes;
- Geração de guias de teste;
- Geração aleatória de falhas, quer seja uma única, quer seja por tempo determinado.

Mesmo assim os objectivos consideram-se cumpridos pois o principal foi provado que é a capacidade do sistema corresponder aos principais objectivos, sendo eles a validação de software e a formação de operadores.

5.2 Trabalho Futuro

Este é um ponto bastante importante para o projecto, pois existe grande interesse por parte do laboratório para a finalização e passagem do projecto para uma versão usável para todo o laboratório. Uma vez que devido aos moldes em que foi desenvolvido o projecto este tornou-se multifacetado, isto é, já não serve apenas para a validação de software de aviação mas também serve para outras plataformas que usem Neptus e Dune, como por exemplo os submarinos a serem desenvolvidos no LSTS.

Assim sendo os próximos passos são o término de todos os casos de uso e objectivos ainda não cumpridos, assim sendo os próximos passos serão no sentido de:

- Criar uma forma de possibilitar a distinção de falhas e erros;
- Terminar o caso de uso Criar guia de teste;
- Possibilitar a criação de falhas e testes aleatórios;
- Possibilitar os restantes tipos de falhas apresentadas nesta tese.

Uma vez cumpridos os objectivos finais o sistema será testado e depois disso integrado nas ferramentas do laboratório para possibilitar o acréscimo de mais uma valia aos sistemas do mesmo. Tornando-os desta forma cada vez mais completos e competitivos.

Referências

[1]. Cloud Cap Technology. Piccolo II Home Page. Recolhido a 28 de Janeiro de 2010. http://www.cloudcaptech.com/piccolo_II.shtm

[2]. Henry, J., Stiff, J. C., Shirar A. J. (2003). Assessing and Improving Testing of Real-time Software using Simulation. Simulation Symposium.

[3]. Peaden, C. J. (2005). Using Simulation for Launch Team Training and Ealuation. Winter Simulation Conference.

[4]. Hildreth, B. L., Jackson, E. B., (2009). Benefits to the Simulation Training Community of a New ANSI Standard for the Exchange of Aero Simulation Models. Interservice/Industry Training Systems Conference, Orlando.

[5]. Koo, C. H., Lee, S. K., (2009).Approach to the use of simulated Software Test Bench in integration test of flight software. Telecommunications Energy Conference.

[6]. Vaglienti, B., Niculescu, M., Hammitt, J., (2009). Piccolo Simulator. http://www.cloudcaptech.com/download/Piccolo/Piccolo%20Documentation/Version%202.1.1%20Docs/Software/Piccolo%20Simulator.pdf

[7]. Robert N. Charette, (2009). Automated to Death. *IEEE Spectrum.* *http://spectrum.ieee.org/computing/software/automated-to-death*

[8]. Kranz, G. (2000). Failure is not an Option: Mission Control from Mercury to Apollo 13 and Beyond.

[9]. P. Dias, R. Gomes, J. Pinto, S. Fraga, G. Gonçalves, J. Sousa, F. Pereira, “Neptus – A Framework to Support Multiple Vehicle Operation“, Proceedings of the Oceans ’05 Europe, Brest, France, June 2005.

[10]. P. Dias, R. Gomes, J. Pinto, S. Fraga, G. Gonçalves, J. Sousa, F. Pereira, “Mission Planning and Specification in the Neptus Framework”, Proc. of the International Conference on Robotics and Automation - ICRA, USA, May 2006

[11]. A. Paiva (2008). TQSO – Teste e Qualidade de Software, Slides da cadeira do Mestrado Integrado em Engenharia Informática e computação.

Índice Remissivo