MODELAGEM E VERIFICAÇÃO FORMAL DE SOFTWARE EMBARCADO ESPACIAL SEGUNDO A NORMA PUS

RODRIGO P. PONTES

Departamento de Engenharia Mecânica, Instituto Tecnológico de Aeronáutica - ITA
Vila Vila das Acácias, 12.228-900, São José dos Campos – SP, Brasil
E-mail: rpastl@ita.br

EMÍLIA VILLANI[1], ANA M. AMBRÓSIO[2]

[1]Departamento de Engenharia Mecânica, Instituto Tecnológico de Aeronáutica – ITA
Vila das Acácias, 12.228-900, São José dos Campos – SP, Brasil
[2]Departamento de Sistemas e Solo, Instituto Nacional de Pesquisas Espaciais – INPE
Avenida dos Astronautas, 1758, CEP 12.227-010, São José dos Campos – SP, Brasil
E-mails: [1]evillani@ita.br, [2]ana@dss.inpe.br

Abstract— This paper discusses the formal verification of an On-Board Data Handling software of satellites based on model checking. Despite the several works on literature, the application of the formal verification techniques for aerospace embedded systems is not a Brazilian reality. In this sense, this paper presents the modeling and formal verification of an On-Board Data Handling software and discusses the main problems and advantages of its utilization. The case study presented in this work is based on PUS (Packet Utilization Standard) proposed by the ECSS (European Cooperation for Space Standardization). The ECSS standards were proposed to standardize the development of space systems in the European Community and these standards have also been adopted in Brazil. The PUS standardizes the communication between the ground system and the on-board system through the definition of a set of services offered by the on-board computer that manages the satellite. Among the PUS services, it is presented as a case of study the Telecommand Verification Service and the On-Board Operations Scheduling Service. The used formal verification is based on timed automata and uses the UPPAAL tool checker to verify the properties of the model. The definition of the properties list to be checked is based on the PUS service description.

Keywords— Timed Automata, ECSS, Model Checking, PUS, On-Board Data Handling

Resumo— Este artigo discute a verificação formal baseada em model checking para software de gerenciamento de bordo de satélites. Apesar dos diversos trabalhos presentes na literatura, a aplicação de técnicas de verificação formal para software embarcado espacial não é uma realidade no contexto nacional. Neste sentido este artigo apresenta a modelagem e verificação formal de software de gerenciamento de bordo e discute os principais problemas e vantagens de sua utilização. O estudo de caso apresentado no artigo é baseado no padrão PUS (Package Utilization Standard), proposto pela ECSS (European Cooperation for Space Standardization). As normas ECSS foram propostas para padronizar o desenvolvimento de sistemas espaciais na Comunidade Europeia e têm sido adotadas também no Brasil. O PUS padroniza a comunicação entre solo e bordo por meio da definição de um conjunto de serviços oferecidos pelo um computador de bordo para gerenciamento do satélite. Dentre os serviços PUS, são apresentados como estudos de caso os serviços "Telecommand Verification Service" e "On-Board Operations Scheduling Service". A verificação formal utilizada é baseada em autômatos temporizados e na ferramenta UPPAAL para verificação de propriedades do modelo. A definição da lista de propriedades a serem verificadas é baseada na descrição do serviço PUS.

Palavras-chave— Autômatos Temporizados, ECSS, Model Checking, PUS, Software de Gerenciamento de bordo

1 Introdução

Este artigo tem como objetivo identificar pontos chaves no desenvolvimento e na verificação formal para duas funcionalidades padronizadas de um software de gerenciamento de bordo aeroespacial. São analisados pontos tais como: tamanho dos modelos, número de requisitos, tempo necessário para o desenvolvimento e verificação das propriedades, entre outros. Essa verificação faz parte de um estudo, cujo objetivo principal é a análise comparativa entre métodos de verificação e validação de software embarcado aeroespacial: métodos formais e testes.

O computador de bordo de um satélite é tradicionalmente considerado um sistema crítico. A manifestação de uma falha pode ocasionar a perda do satélite e da missão espacial.

Nas últimas décadas, a complexidade do software nos computadores de bordo de satélites tem aumentado, no entanto, a atenção dedicada ao seu projeto e verificação não tem sido incrementada na mesma proporção. Leveson (2005) traz uma análise das causas de um conjunto de acidentes espaciais recentes que resultaram na perda de missão. Em todos os casos, software teve alguma participação.

Diferentemente do hardware, que está sujeito a falhas imprevisíveis, uma falha de software é resultado de um erro de projeto. Quando são criadas cópias redundantes do computador de bordo para

aumentar a confiabilidade do sistema, os erros de software também são copiados.

Neste contexto, este trabalho analisa a utilização de autômatos temporizados e *model checking* para verificação de software embarcado de gerenciamento de bordo de satélites, comumente denominado OBDH (On Board Data Handling), mostrando uma comparação entre alguns aspectos entre o desenvolvimento de um serviço de baixa complexidade e um serviço de alta complexidade.

O software de OBDH é padronizado pela ECSS (European Cooperation on Space Standardization), na norma ECSS-E-70-41A, também conhecida como padrão PUS (Packet Utilization Standard). Esta norma define um conjunto de dezesseis serviços para realização da comunicação solo-bordo e gerenciamento de bordo.

As normas ECSS, apesar de constituírem uma proposta da comunidade européia, vêm sendo adotadas pelo INPE (Instituto Nacional de Pesquisas Espaciais) no desenvolvimento de satélites brasileiros. Em particular, o padrão PUS deve ser utilizado em missões futuras, como por exemplo, o satélite ITASAT.

Este artigo está organizado da seguinte forma. A Seção 2 discute trabalhos relacionados. A Seção 3 introduz os autômatos temporizados e a ferramenta UPPAAL, utilizada para modelagem e verificação. A Seção 4 introduz o padrão PUS e detalha os dois serviços utilizados como estudo de caso: *Telecommand Verification Service* e *On-Board Operations Scheduling Service*. A Seção 5 descreve a metodologia utilizada neste estudo, apresenta os modelos desenvolvidos e sua verificação. Por fim, a Seção 6 traz algumas conclusões e discute trabalhos futuros.

2 Trabalhos Relacionados

Esta seção apresenta alguns trabalhos relacionados, visando contextualizar a contribuição deste artigo.

Sreemani e Atlee (1996) analisam a viabilidade de se aplicar *model checking* à especificação de requisitos de software, usando como estudo de caso a especificação de requisitos de software da aeronave militar A-7E. A ferramenta utilizada é o SMV (*Symbolic Model Verifier*). A especificação do software foi fornecida na notação SCR (*Software Cost Reduction*) e traduzida para SMV por meio de uma ferramenta computacional. Foram verificadas cinco propriedades de segurança, uma das quais se demonstrou falsa. Este artigo demonstra que o uso de *model checking* para aplicações críticas reais é viável.

O trabalho de Chan et al (1998) também analisa a aplicabilidade de *model checking* para grandes especificações de software. Este trabalho utiliza como estudo de caso o TCAS II, um sistema de proteção contra colisão no espaço aéreo, de uso obrigatório nos Estados Unidos. Neste caso a especificação do sistema foi fornecida em RSML (*Requirements State Machine Language*) e a ferramenta usada foi SMV. A abordagem proposta para trabalhar com a complexidade do sistema foi uma análise interativa, baseada no refinamento de componentes.

Ogawa et al (2008) abordam o problema de definição das propriedades a serem verificadas. Este trabalho propõe um método para especificar as propriedades baseado na análise dirigida a objetivos (*goal-oriented analysis*) da especificação de requisitos. A ferramenta utilizada como verificador é o SPIN e as propriedades a serem verificadas são especificadas em LTL (*Linear Temporal Logic*).

As principais diferenças entre os trabalhos de Sreemani e Atle, Chan, e este trabalho são que neste trabalho se partiu de uma especificação textual dos requisitos do sistema, onde nenhum modelo formal é fornecido. Também se optou por utilizar a notação de autômatos, esta decisão é baseada na disponibilidade de ferramentas para geração de código automático a partir de modelos em máquinas de estado. Em relação à especificação de propriedades, este trabalho propõe a derivação direta das propriedades a partir dos requisitos. Cada requisito é associado a uma ou mais propriedades a serem submetidas ao verificador.

3 Autômatos Temporizados e a Ferramenta UPPAAL

Esta seção descreve autômatos temporizados e a ferramenta UPPAAL, utilizada para o desenvolvimento dos modelos e sua verificação.

3.1 Autômatos Temporizados

Autômatos temporizados são um formalismo clássico utilizado para modelar sistemas de tempo real. Uma descrição detalhada do formalismo pode ser encontrada em Allur e Dill (1994).

Basicamente, os autômatos temporizados possuem as mesmas características de um autômato comum: um estado inicial; um (ou mais) estado de aceitação; um conjunto finito de transições e um alfabeto de eventos. Também são representados por grafos, onde os estados são os nós, as transições são os arcos e os rótulos desses arcos são os eventos de seu alfabeto. A modelagem tempo é realizada pela definição de um novo tipo de variável: os relógios (*clocks*). Os relógios são variáveis representadas por números reais que modelam o tempo do sistema. Eles evoluem a uma mesma taxa de tempo e só podem ser alterados por meio de uma reinicialização disparada por uma transição.

A escolha dos autômatos temporizados para modelagem se justifica dentro de um contexto mais amplo, cujo objetivo é a comparação com uma

metodologia de testes também baseada na modelagem do sistema como autômatos.

3.2 Model Checking e a Ferramenta UPPAAL

A abordagem de verificação formal aqui utilizada se baseia na modelagem do sistema em autômatos temporizados e no uso do *model checker* UPPAAL (Behrmann et al, 2004).

Model Checking é uma técnica para verificar se um modelo respeita ou não uma determinada propriedade. No caso do UPPAAL, este modelo deve ser especificado como um autômato temporizado, enquanto as propriedades são especificadas num subconjunto da linguagem CTL (Computational Tree Logic). Esta linguagem suporta cinco tipos de propriedades: *possivelmente* (`E<>  p`), *invariantemente* (`A[]  p`), *potencialmente sempre* (`E[]  p`), *eventualmente* (`A<>  p`) e *implica em* (`p -->  q`). Entre as restrições apresentadas, o UPPAAL não permite o aninhamento de operadores.

Como resultado da verificação o *model checker* fornece uma resposta que pode ser propriedade verdadeira ou propriedade falsa. No caso de propriedade falsa, o UPPAAL pode fornecer o caminho que resultou na violação da propriedade. Outra possível resposta é falha de memória, o que indica que o *model checker* não conseguiu obter nenhuma conclusão a respeito da propriedade.

Entre os recursos disponibilizados pelo UPPAAL para especificação do sistema está o uso de redes de autômatos temporizados, que interagem entre si por meio de canais síncronos ou do tipo broadcasting, a associação de variáveis globais e locais aos autômatos, o uso de funções ativadas durante uma transição, a utilização de estruturas de dados padrão e criados pelo usuário, e a instanciação de autômatos.

As justificativas para a escolha do UPPAAL entre as diversas ferramentas disponíveis são o fato de o UPPAAL ser disponibilizado gratuitamente, a sua interface amigável, e a familiaridade dos autores com os recursos oferecidos pela ferramenta.

4 O padrão PUS e o Telecommand Verification Service

Esta seção introduz o padrão PUS e detalha os serviços de "*Telecommand Verification Service*" e "*On-Board Operations Scheduling Service*".

4.1 Padrão PUS

A ECSS (do inglês *European Cooperation for Space Standardization*) é um esforço cooperativo da ESA (Agência Espacial Europeia), agências espaciais nacionais e associações industriais da Europa com o propósito de desenvolver e manter normas comuns. O PUS (*Packet Utilization Standard*) é um dos padrões da ECSS, definido na norma ECSS-E-70-41-A, publicada em janeiro de 2003. O PUS define a interface em nível de aplicação entre solo e bordo, no intuito de satisfazer os requisitos de integração elétrica, testes e operações de vôo. Seus principais benefícios são a redução de custos e de esforços na concepção, desenvolvimento e operação de missões espaciais.

O PUS conta com um conjunto único que totaliza dezesseis serviços, apresentados na Tab.1. Alguns destes serviços são obrigatórios para o OBDH, outros são opcionais. O conjunto de serviços a ser implementado é customizado de acordo com a missão. O PUS também padroniza o campo "*Packet Data Field*" dos pacotes de TC (telecomando) e TM (telemetria).

Tabela 1. Serviços do padrão PUS.

Serviço	Número do Serviço
Telecommand Verification	1
Device Command Distribution	2
Housekeeping and Diagnostic Data Reporting	3
Parameter Statistics Reporting	4
Event Reporting	5
Memory Management	6
Function Management	8
Time Management	9
On-Board Operations Scheduling	11
On-Board Monitoring	12
Large Data Transfer	13
Packet Fowarding Control	14
On-Board Storage and Retrieval	15
Test	17
On-Board Operations Procedure	18
Event-action	19

O *Telecommand Verification Service*, ou serviço de verificação de TC, fornece a capacidade para a verificação de cada execução de um pacote de TC, desde sua aceitação em bordo até a conclusão de sua execução. Apesar de fornecer a possibilidade para a verificação do TC, não existe nenhuma implicação para que todos os TCs de uma dada missão devam ser necessariamente verificáveis em todos os seus estágios. Para muitos TCs o processo de aplicação pode ter pouco (ou nenhum) conhecimento de sua execução.

A verificação de um pacote de TC é dividida em quatro estágios de processamento. São eles:

- **Aceitação do TC;**
- **Início da execução do TC;**
- **Progresso da execução do TC;**
- **Conclusão da execução do TC.**

O serviço deve gerar obrigatoriamente um relato de falha caso um TC falhe em qualquer um dos

estágios mencionados. Caso se queira um relato de sucesso na execução de um desses estágios, será necessária a solicitação deste relato através de um dos campos do pacote deste TC. Os relatos fornecem dados auxiliares para que o sistema de solo tenha total entendimento da execução do TC. Eles podem ser de falha ou de sucesso. O serviço também estabelece um conjunto de relatos que pode ser implementado. Cada estágio deve fornecer um relato de sucesso e um de falha, totalizando oito relatos. Neste trabalho foram implementados o primeiro e o quarto estágios de verificação.

4.3 On-Board Operations Scheduling Service

O *On-Board Operations Scheduling Service*, aqui traduzido como serviço de agendamento de operações em bordo, fornece aos processos de aplicação a capacidade de utilizar os TCs pré-carregados a bordo do satélite e que já foram liberados para a execução. Para isso, o serviço deve manter uma agenda (ou uma fila, aqui chamada de *Command Schedule*) contendo TCs e garantir precisamente a execução destes.

A agenda possui os TCs com seus respectivos atributos de agendamento como: tempo de liberação (quando o TC será liberado e se seu tempo de liberação é relativo a algum evento ou se é absoluto), status de liberação (habilitado ou desabilitado), a qual evento sua liberação é dependente (caso haja dependência), entre outros.

Este serviço deve ter acesso a informações que são vitais para o correto gerenciamento da agenda. São principalmente: o tamanho máximo da agenda, a lista de fontes de onde este serviço pode receber e inserir os TCs e, a lista de processos de aplicação para onde este serviço pode liberar os TCs.

Sempre que o tempo de liberação de um TC for atingido e este não tiver sido liberado, o serviço deve retirá-lo da agenda. A liberação do TC deve ocorrer se, e somente se, a seguinte condição for satisfeita: o seu status de liberação for "habilitado", sua execução está desbloqueada devido à dependência a algum evento; e, o destino do TC está habilitado para executá-lo.

O usuário pode requisitar do serviço um total de dezenove atividades, cada uma com seu respectivo subtipo de identificação. Para este artigo apenas os subtipos da Tab.2 foram considerados.

Tabela 2. Atividades implementadas.

Atividade	Número do Subtipo
Habilitar Liberação de TCs	1
Desabilitar Liberação de TCs	2
Reset da Agenda	3
Inserir TCs na Agenda	4
Retirar TCs da Agenda	5
Time-Shift de todos os TCs	15

5 Estudo de Caso

5.1 Metodologia

A metodologia utilizada neste trabalho é apresentada na Fig.1.

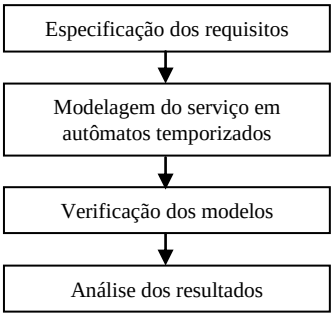

Figura 1. Metodologia.

A partir da descrição de cada serviço presente na norma ECSS-E-70-41A, foi elaborada uma especificação de requisitos na forma textual. Foi então elaborado um modelo em autômatos temporizados para cada um dos serviços considerados. Inicialmente, os modelos construídos foram analisados por meio de simulação. Os erros detectados foram corrigidos.

A verificação formal dos modelos foi realizada associando a cada requisito uma ou mais propriedades em CTL, que foram submetidas ao *model checker* do UPPAAL.

Os erros detectados foram corrigidos e subsidiaram a análise da contribuição do processo de *model checking* para a especificação do sistema.

5.2 Elaboração do documento de requisitos

O documento de requisitos gerado para os serviços nas seções 4.2 e 4.3 contém um total de 123 requisitos, dos quais 19 são para o *Telecommand Verification Service* e 104 são para o *On-Board Operations Scheduling Service*.

Como um primeiro exemplo, são apresentados três requisitos para o primeiro estágio de verificação do *Telecommand Verification Service*.

R4 - A verificação do telecomando recebido deve ser realizada antes da execução deste telecomando.

R4.2 – O relato de sucesso só deve ser gerado se o primeiro bit do campo "Ack" do pacote de telecomando estiver em 1.

R4.2.1 – O relato de sucesso de aceitação deve ser de subtipo 1.

Como outro exemplo, são apresentados três requisitos de propósito geral para o *On-Board Operations Scheduling Service:*

R11 – Todos os pedidos e relatos do On-Board Operations Scheduling Service devem ser de tipo de serviço igual a 11.

R18 – No estado inicial, o Command Schedule deve estar vazio.

R19 – No estado inicial, o Command Schedule deve estar desabilitado.

5.3 Modelagem em Autômatos Temporizados

Para o modelo do *Telecommand Verification Service* foram desenvolvidas 6 classes. São elas: *Solo_TC(), Solo_TM(), Pac_TC(), Pac_TM(), OBDH()* e *Processo()*. As duas primeiras classes representam a estação que envia o pacote de TC e que recebe o pacote de TM, respectivamente. As classes *Pac_TC* e *Pac_TM* são uma representação dos pacotes de TC e TM, que, no primeiro caso, são enviados pelo sistema de solo e, no segundo caso, são recebidos pelo sistema de solo. Os pacotes de TC e TM foram implementados como estruturas de dados *Telecomando* e *Telemetria*, respectivamente. Todos os canais de comunicação desse modelo são do tipo *channel*, o que faz com que dois autômatos, sincronizados entre si, só evoluam de estados quando ambos possuem a mesma transição.

A classe *OBDH* representa o computador de bordo, o qual contém a implementação deste serviço, conforme a Fig. 2. Por fim, a classe *Processo* representa o processo de aplicação ao qual será destinado o pacote de TC, recebido e inicialmente verificado pelo OBDH.

A operação ocorre da seguinte maneira: o sistema de solo monta um pacote de TC e envia ao computador de bordo. Este envio é representado pelo evento "Envia_TC" (transição entre os estados "Sem_Novo_TC" e "Próximo_Estágio"). Nesta mesma transição ocorre a atualização da variável "tc_busy", indicando que o *Telecommand Verification Service* está ocupado. É através desta variável que o *On-Board Operations Scheduling Service* identifica se o serviço de verificação de telecomandos está apto a receber algum TC proveniente da *Command Schedule.*

Ao receber o TC, o *Telecommand Verification Service* faz, primeiramente (primeiro estágio representado pela variável TC_P=0), a verificação dos campos do pacote recebido, procurando por alguma eventual anormalidade devido à recepção do pacote. Em seguida, é realizada a verificação de alguns pontos chaves como: identificação de um processo de aplicação inexistente; tamanho do pacote ilegal; *checksum* do pacote incorreto; tipo ou subtipo de serviço inexistente no OBDH; ou dados contidos incorretos. Isso é realizado pela função "tc()", na transição do estado "Próximo_Estágio" para "Verificando".

No caso de não haver falhas no pacote de TC, ele é enviado ao seu processo destino ou para o *On-Board Scheduling Service*. O serviço verificação fica então na espera da confirmação do recebimento do pacote de TC por parte da aplicação de destino ou por parte do *On-Board Operations Scheduling Service*.

No momento do recebimento, o processo informa ao serviço se o TC foi aceito ou não. No caso de não ter sido aceito, é enviado um relato ao sistema de solo. Quando o TC é aceito, só é enviado o relato se o bit "Telecomando.Ack.b3" for igual a 1. O processo então executa esse TC e informa ao serviço o sucesso ou a falha da execução do TC. O relato de sucesso é apenas enviado quando o bit "Telecomando.Ack.b0" for 1. No caso de falha, o relato dessa falha é obrigatoriamente enviado ao sistema de solo.

Para ilustrar o funcionamento deste serviço, toma-se como exemplo um TC com seu tipo igual a '3'. Quando no estado "Sem_TC", o serviço de verificação recebe o TC do *Solo_TC()* e passa para o estado "Próximo_estágio". Como a variável "TC_P" é igual a '0' (o TC acabou de ser recebido), o serviço passa para o estado "Verificando", já que, na transição entre os dois estados, existe a guarda "TC_P==0". Neste estado é feita a verificação e constatado que existe falha no pacote. Então a variável booleana "falha" é atualizada com o valor '1', indicando que há falha no pacote, e a variável "error_code" é atualizada com o valor '3', informando que o erro ocorreu devido ao tipo do TC não existir na implementação (apenas os tipos '2' e '11' foram considerados neste trabalho). O serviço passa para o estado "Falha_no_Pacote" e, devido ao valor do seu código de erro (a guarda para este estado é "error_code==3"), é passado para o estado "Tipo_de_Pacote_Ilegal".

O serviço vai, imediatamente, para o estado "Pronto_para_Montar_TM", atualizando a variável "TC_P" com o valor zero, para que o pacote não seja verificado novamente. O serviço monta a TM de falha (que deve ser obrigatoriamente enviada ao sistema de solo) informando o código de erro e passa para o estado "Pacote_TM_Montado". Rapidamente, o serviço envia ao *Solo_TM()* este pacote de TM e vai para o estado "Decisão_próximo_TC". Neste estado é visto se o pacote vai para o próximo estágio de verificação (no caso de ter sido aceito e ainda não executado), ou se pacote vai ser descartado (no caso de falha ou conclusão de sua execução). No caso do exemplo dado, o serviço descarta o pacote e vai para o estado "Sem_Novo_TC", atualizando a variável "tc_busy" com o valor '0', informando que o serviço está desocupado.

Para o *On-Board Operations Scheduling Service* foram também desenvolvidas 6 classes: *OBDH_OOSS()* , *Ground_System_Sender(), Ground_System_Receiver(), TC_Verification(), Queue_TC(), Telecommand().* A primeira classe representa o gerenciador do Command Schedule, ou fila de TCs. O Command Schedule é representado pela classe *Queue_TC().*

As classes *Ground_System_Sender()* e *Ground_System_Receiver()* representam a estação de solo no momento de envio do pacote de TC e recepção do pacote de TM, respectivamente. A classe *Telecommand()* representa o TC recebido pelo computador de bordo e mostra seu conteúdo; e, por último, a classe *TC_Verification()* representa uma implementação do *Telecommand Verification Service*.

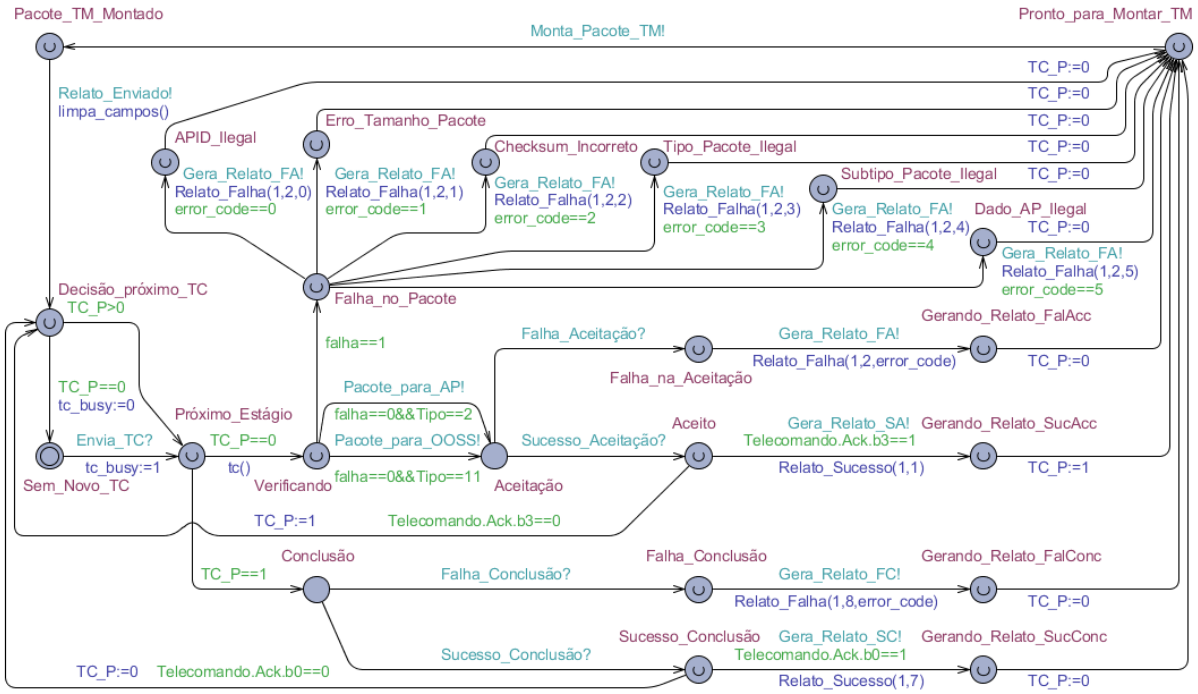

Figura 2. Modelo do OBDH para o Telecommand Verification Service.

Da mesma maneira que o *Telecommand Verification Service*, primeiramente é montado um pacote de TC pelo sistema de solo e então enviado ao OBDH, que recebe e faz a verificação do TC e passa para o *On-Board Operations Scheduling Service*, onde este fará o gerenciamento do TC recebido. Este gerenciamento é feito de acordo com o subtipo válido contido no TC. Os subtipos válidos são os mesmos da Tab.1 da seção 4.1.

A atividade mais crítica deste serviço é a inserção de um TC no Command Schedule, pois se deve haver um cuidado maior para o correto gerenciamento de seu funcionamento. Os TCs são organizados em ordem crescente através de seus tempos de liberação de execução. À medida que um TC é liberado para execução, ele é retirado do Command Schedule e este é reorganizado. Aqui se faz o uso de variáveis *clock* para modelar o tempo de liberação, mostrando o funcionamento deste serviço.

5.4 Verificação de Propriedades

Como pode ser visto em Pontes et al (2009), a verificação do modelo é organizada em três atividades:

1. Simulação, o qual engloba simulação aleatória e cenários específicos. Este primeiro passo detecta a maioria dos erros do modelo.
2. Verificação de propriedades esperadas que não estejam especificamente relacionadas aos requisitos, como a ausência de deadlock e alcançabilidade de estados chaves.
3. Verificação dos requisitos, i.e, a definição das propriedades em fórmulas CTL que estão relacionadas aos requisitos e sua verificação.

Entre esses três passos, o mais crítico é o último. Os requisitos são definidos em linguagem informal e não há regras para traduzir em fórmulas CTL. As seguintes abordagens foram utilizadas:

1) O requisito pode ser traduzido diretamente para uma ou mais fórmulas CTL. A verificação do requisito é a prova das formulas CTL.
2) O requisito é verificado por inspeção do modelo.
3) A verificação do requisito é decomposto na prova da fórmula CTL e na inspeção visual do modelo.

Para ilustrar este processo, apresentam-se alguns exemplos, a começar pelo requisito R4.2 (Seção 5.2). Este afirma que o computador de bordo deve gerar um relato de sucesso na aceitação apenas quando um bit específico de um dos campos do TC estiver com o valor ‘1’. Para definir as propriedades para este requisito, é necessário pensar em todos os caminhos possíveis para acessar o estado onde se cria este relato. Primeiramente, notou-se, por inspeção da classe da Fig.2, que há apenas um único caminho tomado para o estado “Gerando_Relato_SucAcc” e que, necessariamente, tem que passar pela condição de guarda “Telecomando.Ack.b3==1”, que é o bit especificado. Para elucidar quaisquer dúvidas, foram propostas 2 propriedades em linguagem formal. Por ser um bit, o “Telecomando.Ack.b3” só pode assumir dois valores: ‘0’ ou ‘1’. Portanto, as propriedades são:

P1– Estar no estado “Gerando_Relato_SucAcc” implica que o bit 3 do campo “Ack” está obrigatoriamente em 1. Fórmula CTL:

A[] OBDH_1.Gerando_Relato_SucAcc imply Telecomando.Ack.b3==1

P2– Estar no estado "Gerando_Relato_SucAcc" implica que o bit 3 do campo "Ack"não está 0. Formula CTL:

A[] OBDH_1.Gerando_Relato_SucAcc imply not Telecomando.Ack.b3==0

O restante das propriedades CTL foi verificado utilizando o mesmo padrão de universalidade "A[] p imply q", que significa que "p" sempre implica em "q".

Para a verificação dos requisitos R18 e R19 do *Scheduling Service*, foram utilizados a definição de propriedades e inspeção do modelo. Para essa verificação, as propriedades abaixo foram consideradas:

P3 – Quando o Command Schedule *é iniciado, não existe nenhum TC contido nele:*

A[] Queue_1.Empty_Queue imply i==0

A variável "i" indica a quantidade de TCs contida no Command Schedule. Em seu estado inicial, por inspeção, nota-se a ausência de TCs contidos nele.

P4 – Quando o Command Schedule *é iniciado, seu status tem o valor zero.* Fórmula CTL:

A[] OBDH.Disabled_Schedule_Waiting_TC imply status==0

Além disso, ao se inspecionar o modelo, percebe-se que no seu estado inicial, o *Command Schedule* está desabilitado.

5.5 Análise dos resultados

A Tabela 3 apresenta alguns valores para a comparação do esforço requerido nas atividades de modelagem e verificação formal dos dois serviços. Esta tabela também inclui o número de erros encontrados por meio da verificação. É importante observar que, para os dois serviços, a modelagem e verificação dos requisitos foram realizadas pela mesma pessoa, que já tinha conhecimento prévio da ferramenta UPPAAL.

Tabela 3. Informações relevantes da modelagem.

	Telecommand Verification Service	On-Board Operations Scheduling Service
Nº de requisitos	19	104
Nº de estados	35	61
Nº de transições	54	149
Horas para modelagem	46	103
Nº de propriedades	20	26
Horas para verificação	4	15
Nº erros encontrados	13	4

O número de requisitos está relacionado à complexidade do serviço. Pode-se notar que à medida que a complexidade do serviço aumenta, aumentam as quantidades de estado e transição. No entanto, esta evolução não é necessariamente proporcional. Isto porque o serviço de *scheduling* inclui a manipulação de estruturas de dados, que resulta em muitos requisitos para a sua definição, mas pode ser modelada em autômatos com poucos estados. Se considerarmos a média do número de estados e transições, observa-se que esta média é aproximadamente proporcional ao número de horas despendidas para modelagem.

O número de propriedades também não está diretamente relacionado ao número de requisitos. Verifica-se neste trabalho que a maioria dos requisitos do *On-Board Operations Scheduling Service* diz respeito às estruturas de dados que devem ser utilizadas e, portanto, podem ser verificados por inspeção dos dados que foram alterados e trabalhados nos modelos.

Para o caso do serviço de *scheduling*, destacaram-se três erros principais. O primeiro foi o de atualização da variável *status*. Neste trabalho esta variável é utilizada para indicar o atual status de liberação dos TCs. Caso esse indicador de status esteja desatualizado, pode acarretar no mau funcionamento do gerenciador, fazendo-o liberar ou não os TCs em momentos inoportunos, o que pode, possivelmente, levar a uma falha grave na missão. Este foi considerado um erro crítico e se deu no processo de modelagem.

O segundo erro foi o de descarte de um TC. Ao se abrir e utilizar um TC, verificando que havia um erro no subtipo do pedido, o sistema não passava a devida informação para o serviço de verificação de TC, que ficava na espera dessa informação, o que terminou no bloqueio de todo o sistema, perdendo assim, a missão. Portanto, deve-se fazer o correto cadastramento dos tipos e subtipos dos serviços implementados a fim de evitar tais problemas. Este também foi considerado um erro crítico e se deu à falta dessa informação no documento de requisitos, proveniente da norma.

O terceiro foi a explosão de estados durante a verificação de ausência de bloqueio do sistema. Deve-se tomar cuidado no intervalo de valores de variáveis que são importantes em um TC. Um exemplo é o número de identificação do TC. Este, por possuir de tamanho máximo de dois bytes, pode assumir valores reais (sem sinal) entre '0' e '65535' (16 bits), aumentando sensivelmente o número de estados do sistema. Este na verdade não foi um erro de modelagem, mas sim um resultado da limitação do processo de *model checking*.

Para o *Telecommand Verification Service*, poucos foram os erros críticos. O primeiro erro foi no estágio de aceitação. O computador de bordo assumia que o processo destino sempre aceitava qualquer TC. Este erro foi considerado de criticidade média. Outro

erro, de baixa criticidade, foi que, mesmo que o usuário do sistema escolhesse não receber o relato de sucesso na aceitação do pacote de TC, ele o receberia de qualquer maneira. Outro erro de criticidade média foi a inversão dos bits do campo "Ack" do pacote de TC. Os erros encontrados neste serviço foram devido ao processo de modelagem e não comprometem a missão.

Outra contribuição do processo de *model checking* se refere à revisão da especificação de requisitos. Foi observado que alguns requisitos não foram escritos de maneira clara, de forma a possibilitar a definição da propriedade. Desta forma, o processo de verificação forma contribuiu também para a revisão dos requisitos.

6 Conclusão

Este trabalho apresentou um estudo de caso relativo à aplicação de técnicas de verificação formal baseadas em autômatos temporizados no desenvolvimento de software embarcado de gerenciamento de bordo para satélites. Foram modelados e verificados dois serviços do total de dezesseis serviços do padrão PUS, que caracteriza o comportamento do software de gerenciamento de bordo.

Os resultados obtidos permitem concluir que a verificação formal é viável e contribui não apenas para a detecção de erros como também para revisão da especificação de requisitos do sistema. Alguns dos erros detectados nesta etapa foram considerados erros críticos, provenientes tanto do processo de modelagem quanto da especificação dos requisitos a partir do padrão PUS.

Particularmente, no que se refere à implementação do padrão PUS, este estudo de caso aponta possíveis pontos de ambigüidade da norma, ou pontos mais sensíveis à interpretação errônea. Além disso, alguns pontos são passíveis de múltiplas interpretações.

Observou-se que o esforço de modelagem é proporcional ao tamanho do modelo, mas não está diretamente relacionado com o número de requisitos. O esforço para a verificação mostrou-se proporcional ao número de requisitos, apesar do número de propriedades especificadas não estar relacionado ao número de requisitos.

Os pontos frágeis do processo de verificação formal são (1) a definição do conjunto de propriedades que verificam os requisitos textuais, e (2) a conversão do modelo para o código. Enquanto esta conversão pode ser coberta por ferramentas de geração automática de código, a definição das propriedades é um passo inerentemente informal e suscetível a erro ou incompleteza.

Neste sentido, este trabalho é parte de um escopo mais amplo onde se pretende verificar os erros inseridos nestas duas etapas, e, portanto 'sobrevivente' ao processo de *model checking*. Esta análise será realizada por meio da aplicação de testes no produto final, isto é, no código gerado a partir do modelo UPPAAL.

Quanto a definição das propriedades, entre os trabalho futuros está a aplicação do conceito de *patterns* (Dwyer et al, 1999) na especificação de requisitos, como forma de direcionar a conversão dos requisitos em propriedades CTL.

Agradecimentos

Esta pesquisa é financiada pelo projeto FINEP/CTA/INPE Sistemas Inerciais para Aplicação Aeroespacial (SIA).

Referências Bibliográficas

Allur R. and Dill D. A. (1994) Theory of Timed Automata. Theoretical Computer Science, v. 126, p. 183-235.

Behrmann, G. et al (2004), A Tutorial on Uppaal. In: Proceedings of the 4th International School on Formal Methods for the Design of Computer, Communication, and Software Systems (SFM-RT'04). LNCS v. 3185.

Chan, W. et al (1998), Model Check Large Software Specifications. Disponível online em: < http://portal.acm.org/citation.cfm?id=250707.239127>. Acesso em: 2 mar 2010.

Dwyer, M.B., Avrunin, G.S., Corbett, J.C. (1999) Patterns in Property Specifications for Finite-state Verification. In: 21st Int. Conf. on Software Engineering.

ECSS (2003), European Cooperation for Space Standardization (ECSS). ECSS-E-70-41A – Ground systems and operations: telemetry and telecommand Packet Utilization. Noordwijk: ESA publication Division.

Leveson, N. (2005), "Role of Software in Spacecraft Accidents". *Journal of Spacecrafts and Rockets*, vol. 41, no. 4, pp. 564-575.

Ogawa, H., Kumeno, F., Honiden, S. (2008), Model Checking Process with Goal Oriented Requirement Analysis. In: 15th Asian-Pacific Software Engineering Conference, p. 377-384.

Pontes, R. P. et al (2009), A Comparative Analysis of two Verification Techniques for DEDS: Model Checking *versus* Model-based Testing. In: Proceedings of 4th IFAC Workshop on Discrete Event System Design (DEDes), p.70-75.

Srremani, T., Atlee, J. M. (1996), Feasibility of Model Checking Software Requirements: A Case Study.