DESENVOLVIMENTO DE SISTEMAS EMBARCADOS PARA APLICAÇÕES ESPACIAIS

Samoel Mirachi, samoel@gmail.com[1, 2]
Francisco das Chagas Carvalho, fchagas.carvalho@gmail.com[1]
[1]Orbital Engenharia LTDA, São José dos Campos, Brasil
[2]Instituto Tecnológico de Aeronáutica-ITA, São José dos Campos, Brasil

Abstract. *This article discusses the development of software for real-time embedded systems in space applications. To conform to space applications, this will be a Critical Real-Time System (hard RTS), operating in a real-time operating system. The design follows the methodology of cascade life-cycle, is modeled in UML 2, developed in ANSI C language (using conception OO in C) and embedded in microcontroller/microprocessor on-board computers. It calculates the control law, perform the on-board data handling and management, telecomand/telemetry with EGSE.*

Keywords: *UML 2, Sistemas embarcados espaciais, Sistema de tempo real crítico, RTOS.*

1. INTRODUÇÃO

Com o advento do uso de sistemas embarcados, hoje tornou comum encontrar aplicações em diversas áreas, como sistemas automotivos (controle de motor, sistemas de frenagem anti-derrapagem), periféricos computacionais (impressoras, scanners), robôs (rampa de controle), aeronaves (sistemas de gerenciamento de vôo) e até espaciais (controle de Velocidade angular e gerenciamento de dados) [1] [2]. Com o desenvolvimento da tecnologia de circuitos integrados e microprocessadores, os sistemas embarcados têm se tornado cada vez mais sofisticados e complexos. Como conseqüência, o processo de desenvolvimento de *software* embarcado também se torna uma atividade custosa e propensa a erros [2].

Este artigo aborda o desenvolvimento de *software* de tempo real para sistemas embarcados em aplicações espaciais, tendo seus diagramas modelados para uma aplicação especifica de uma plataforma suborbital de microgravidade, possuindo como principais funções calcular a lei de controle, gerenciamento de bordo de dados de serviço e de carga útil e telemetria/telecomando com o *Electrical Ground Support Equipment* (EGSE).

2. MATERIAIS E MÉTODOS

Os métodos para o desenvolvimento de sistemas embarcados utilizado neste artigo contemplam uma metodologia para Sistema de Tempo Real Critico (*hard* RTS), integrado a sistema operacional de tempo real (RTOS), seguindo o paradigma de ciclo de vida em cascata, modelado em UML 2, codificado na linguagem ANSI C (utilizando o conceito OO em C), o qual poderá ser embarcado em microcontrolador/microprocessador de diversas plataformas.

2.1. Sistema Operacional de Tempo Real (RTOS)

Um RTOS é um Sistema Operacional destinado à execução de múltiplas tarefas onde o tempo de resposta a um evento (externo ou interno) é pré-definido (prazo), possui recursos de gerenciamento multitarefa em tempo real com escalonamento preemptivo, tratamento de interrupções para RTS, facilidades para comunicação entre processos e sincronização, é apto para iniciação de recursos de hardware, possui mecanismo para prover facilidades de comunicação serial, interface com o desenvolvedor, interface com compilador C compatível com o ANSI C, capacidades de depuração de código e monitoração de desempenho. Para um sistema de tempo real sem RTOS seria necessário o programador construir um *Kernel* para o gerenciamento de escalonamento preemptivo em aplicações de tempo real, fazendo uso do *Generic Fixed Priorities* (FP) e do *Earliest Deadline First* (EDF), por exemplo.

2.2. Sistema de Tempo Real

Para o desenvolvimento deste Sistema em Tempo Real (RTS), optou-se por uso de um RTOS, com o qual se obteve maior confiabilidade e menor tempo de desenvolvimento. Existem dois gêneros de RTS, o rígido (hard RTS) e moderado (soft RTS), o fator que distingue se um *software* é rígido ou moderado é a severidade da pena pelo não cumprimento das tarefas num determinado intervalo de tempo. Com isso o Software embarcado para sistemas espaciais é considerado um hard RTS. Os RTS rígidos são inflexíveis, pois o prazo da tarefa (deadline) não pode ser ultrapassado [1] [7] [11].

2.3. Ciclo de Vida de *Softwares* Embarcados

O processo de desenvolvimento do *software* embarcado utilizará paradigma de ciclo de vida em cascata (*waterfall*), estruturado em cascatas de fases, como ilustra a Figura 1 onde o final de uma fase implica no início de outra. Este tipo de comportamento ressalta a qualidade de um modelo rígido e linear para o desenvolvimento de Software embarcado de tempo real no sentido que uma fase começa após a outra numa direção linear [3].

Para que o *Software* Embarcado tenha uma maior flexibilidade, caso seja necessário, a arquitetura do *Software* Embarcado poderá fazer uso do Modelo em Cascata Revisto, o qual prevê a possibilidade de, a qualquer tarefa do ciclo, regressar a uma tarefa anterior de forma a contemplar alterações funcionais ou técnicas, caso a Coordenação dos Softwares Embarcados venha requerer [3]. As fases e processos do ciclo de vida do desenvolvimento do *Software* Embarcado são apresentados na Figura 1.

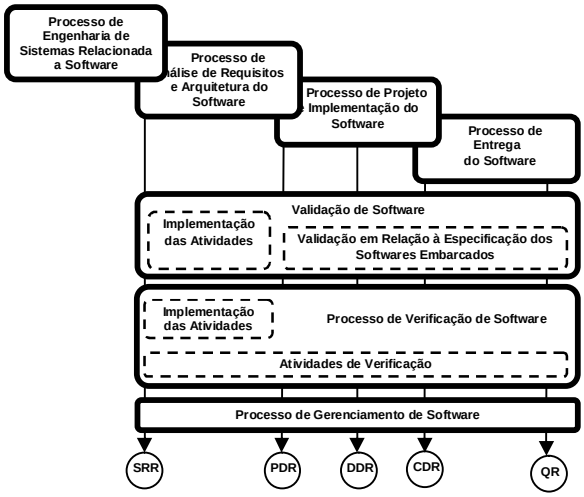

Figura 1. Processo do ciclo de vida do *software* embarcado (ECSS, [3])

2.4. Gerenciamento de Riscos

O objetivo do gerenciamento de riscos em Softwares de tempo real compreende sua identificação e a geração de requisitos de confiabilidade para lidar com estes riscos.

O processo iterativo para a identificação preliminar de cenário de risco envolve a compreensão dos riscos enfrentados pelos Softwares, a descoberta das causas de origem e a geração de ações para gerenciar e mitigar os eventuais riscos [4].

A Tabela 1 apresenta os cenários de riscos e perigos identificados para a aquisição e desenvolvimento dos Softwares embarcados, seus tópicos foram elaborados de acordo com as seguintes definições:

1. Cenários de Riscos: Foram identificados os potenciais riscos que podem ocorrer, tanto para a aquisição de *software*s e ferramentas de desenvolvimento, quanto para desenvolvimento e testes dos *software*s embarcados;
2. Classificação de Riscos: Foi dividida entre sua gravidade e sua probabilidade de ocorrência, ponde ser de três níveis: Alta, Média e Baixa;
3. Impacto de Ocorrência: Mostra o quanto um risco pode acarretar de atraso na entrega da versão final do Software.

Ação Sugerida: Foram preliminarmente identificados os cenários de risco e sugeridas às ações para mitigá-los.

Tabela 1 Identificação Preliminar de Cenários de Risco

Nº	Cenários de Riscos	Gravidade do Risco	Prob. de Ocorrência	Impacto da Ocorrência	Ação Sugerida
1	Faltar requisitos	Alta	Baixa	Atraso na entrega do Produto	Contratar consultor para Revisar os requisitos da Especificação dos *Software*s Embarcados
2	Requisitos incompletos	Alta	Baixa	Produto não atenderá às necessidades do cliente	Realizar reuniões adicionais com o cliente e com a equipe do projeto.
3	Falta de treinamento no RTOS, ferramenta CASE	Alta	Média	Atraso na entrega e/ou inserção de erros no produto	Prover meios para viabilizar o treinamento

Nº	Cenários de Riscos	Gravidade do Risco	Prob. de Ocorrência	Impacto da Ocorrência	Ação Sugerida
4	Indisponibilidade de uso do Ambiente de Testes	Alta	Média	Impossibilidade de realizar os testes de qualificação	Solicitar a aprovação prévia da empresa responsável pelo ambiente de teste.
5	Não realização de todos os testes necessários para a qualificação do SW	Alta	Média	Entrega do produto com possível defeito	Assessoria ou terceirização para o planejamento e execução dos testes do *Software*
6	Falta de equipe mínima necessária para o desenvolvimento, V & V do *software*	Alta	Média	Realização Parcial de Algumas atividades propostas	Contratar consultor ou incorporação de novos servidores para o desenvolvimento e validação do projeto
7	Não recebimento da ferramenta CASE na fase de Engenharia de Requisitos e Arquitetura	Média	Alta	Atraso na entrega do Produto e do documento do *Software*	Prover meios para utilização de ferramentas *open-source* e tentar viabilizar a aquisição destas ferramentas

2.5. Plano de Verificação e Validação

Um programa de verificação e validação (V & V) será estabelecido e implementado através do Plano de Verificação e Validação pela equipe de desenvolvimento e teste do *software*, preparado de acordo com as especificações de sistema, também, em conformidade com os documentos aplicáveis e com os documentos de referência, considerando os métodos de verificação, níveis, modelos e estratégias, para prover um alto nível de verificação para todas as funcionalidades do *software* embarcado [4] e de tarefas de tempo real, utilizando ferramentas como LabView, Matlab e Simulink.

Os seguintes métodos de V & V deverão ser usados para validar a verificação do *software*:

- Análise estática;
- Inspeção;
- Teste/Ensaio.

2.5.1. Análise Estática

A Análise Estática deverá complementar os recursos de detecção de erros providos pelo compilador. Através de técnicas de heurísticas, a análise estática pode descobrir alguns defeitos do programa resultante do uso de compiladores C/C++, esta técnica de V & V é extremamente importante quando C (e, em menor extensão, C++) for usado para desenvolvimento de sistemas críticos. Nesse caso, a análise estática pode descobrir um grande número de erros potenciais, e reduzir significativamente os custos de teste [4].

2.5.2. Inspeção

Técnicas de inspeção deverão ser usadas para verificar a conformidade com requisitos específicos, no qual um *software* é revisto para se encontrar erros, omissões e anomalias (defeitos de programa), tanto unicamente aplicados quanto combinados com testes em todos os estágios e níveis de V & V de projeto e requisitos de construção. Geralmente as inspeções focam o código-fonte, porém qualquer representação legível do *software*, seus requisitos ou modelo de projeto, podem ser inspecionados, trazendo com isso uma grande vantagem se comparado a V & V por teste, não de forma concorrente, mas sim complementar reduzindo com isso o custo de tempo de desenvolvimento [4].

2.5.3. Teste/Ensaio

Quando requisitos forem verificados por medição de desempenho e funcionalidade de equipamentos de subsistemas sob simulação de vários ambientes, o método será referido como “Teste”. O teste é um processo voltado a atingir a confiabilidade do *software* [4].

3. RESULTADOS E DISCUSSÕES

O desenvolvimento do *Software* Embarcado esta sendo baseado em Sistema de Tempo Real Crítico (hard RTS), seu projeto segue a metodologia de ciclo de vida em cascata, sendo modelado em UML 2, desenvolvido na linguagem ANSI C/C++ e embarcado junto a um Sistema Operacional de Tempo Real no computador de bordo.

3.1. Arquitetura do Software Embarcado no Processo de Desenvolvimento de Sistemas

O projeto do *Software* Embarcado está baseado no processo clássico de desenvolvimento de sistemas chamado *waterfall* (cascata), com uma seqüência de fases bem definidas, e com a possibilidade de regressar a uma fase anterior de forma a contemplar alterações funcionais ou técnicas (Modelo em Cascata Revisto) [4] [5].

A arquitetura de *software* para o *software* embarcado seguirá a metodologia de desenvolvimento sendo modelada em UML 2.0. A definição da Arquitetura do Software encontra-se finalizada nesta etapa do projeto. Mas, por se tratar de uma arquitetura flexível, não significa que alterações não possam ocorrer neste modelo. Deste modo, o detalhamento obtido até o momento já é suficiente para prosseguir com o projeto do *software*. Esta arquitetura está sendo construída objetivando todo o sistema e abrangendo os elementos mínimos necessários para implementar a primeira versão do *software* [6].

Os elementos mínimos para o desenvolvimento da primeira versão do *software* devem contemplar a propriedade de flexibilidade para incorporar as novidades que surgirão com a próxima versão. A cada versão a arquitetura é incrementada, mas sem ferir as versões anteriores [2].

Problemas complexos precisam ser entendidos como um todo e por isso são particionados em "partes" que podem ser mais facilmente gerenciadas e detalhadas (representadas nesse documento pelos diagramas de atividades). A composição do todo é feita estabelecendo-se interfaces entre as "partes" que são bem definidas no diagrama de caso de uso. A evolução do *Software* Embarcado e a mudança entre versões esta diretamente relacionada à verificação e validação do sistema através da execução de testes, sendo importantíssima para garantir não só a qualidade do *Software* Embarcado, mas também da arquitetura. Os erros e as falhas a serem apontados no *Software* Embarcado indicam situações de melhoria que serão implementadas nas próximas versões [7] [11].

A conformidade dos requisitos com a arquitetura é constantemente avaliada e verificada. A cada fase que se avança, o registro das não-conformidades servem de subsídios para a melhoria ou evolução da arquitetura atual, provando que a definição e manutenção da arquitetura é um processo incremental e iterativo [6].

3.2. Modelagem Geral do Software Embarcado

Uma linguagem de Modelagem é necessária para descrever a base da construção conceitual do *software*. Dentre os objetivos da linguagem, podem ser destacadas a simplicidade, a padronização, a facilidade de projeto e a argumentação. Este objetivo leva ao desenvolvimento de sintaxe, semântica, documentação e suporte para citação de ferramentas que auxiliam a programação [6].

Entre as vantagens da modelagem, destaca-se a possibilidade de analisar as funcionalidades do *software* já modelado através da simples visualização de seus diagramas, uma vez que o conjunto deles forma um cenário completo do *software*.

Com uso de ferramentas *Computer-Aided Software Engineering* (CASE) para implementação da modelagem do *software*, consegue-se simular e até mesmo, verificar se o *software* atende aos requisitos antes mesmo de sua codificação. Além disso, o padrão UML trás facilidades para análise e possíveis modificações futuras, melhorando com isso a manutenção [2]. Os diagramas UML do *Software* Embarcado estão sendo desenvolvidos com uso da ferramenta IDE *Netbeans* 6.1, que possui características próprias como, por exemplo, padrões de cores e entidades, mas não fugindo do padrão proposto pela UML [5] [8].

3.3. Diagramas em UML

A UML propõe que a modelagem seja realizada através de vários diagramas que facilita a visualização das características do *software* modelado. Para este *software* embarcado foram aplicados cinco diagramas, cada qual com sua função que em conjunto possibilitam o entendimento do funcionamento do *software* como um todo. Estes diagramas são apresentados como comportamentais [8], conforme a Figura 2.

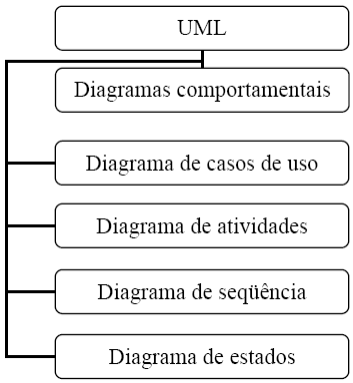

Figura 2. Diagramas Propostos para o Software

3.3.1. Diagrama de Casos de uso do Software Embarcado

O primeiro passo para Modelagem em UML é a elaboração de um Diagrama de Caso de Uso. O propósito deste diagrama é fornecer uma visão global e serviços em alto nível a serem supridos pelo sistema, as relações entre eles e os atores envolvidos. O ícone do estereótipo ator é representado pela figura de um individuo [5], podendo ser quaisquer entes físicos, outros *software*s ou humanos que interagem diretamente com o *software*, [2].

Definindo-se os atores, o próximo passo é definir os casos de uso, representados graficamente mediante uma oval com uma frase que representa uma determinada ação [5], que são as funcionalidades que o sistema deve fornecer, cada caso de uso representa uma função do Software Embarcado, ou seja, uma capacidade do sistema nomeada. Cada caso de uso deve ser documentado com o fluxo de evento a ser executado. Este fluxo mostra os eventos que iniciam que devem ser executados e que finalizam o caso de uso.

Então através deste diagrama é possível verificar quais são os atores que interagem com o *software* e as funcionalidades que ele dever fornecer. As Figura 3 e Figura 4 apresentam os Diagramas de caso de uso do Software Embarcado. O caso de uso "gerenciamento de bordo" e o caso de uso "controlador de bordo", que são as principais tarefas do Software Embarcado, são executados através de várias outras tarefas detalhadas (processar sinais e dados, gerar sinal de controle, executar o Watchdog, comutação de energia, processar os telecomandos e processar as telemetrias), os atores são os que enviam sinais de entrada para o Software Embarcado. Eles enviam seus respectivos dados para o conversor de sinais de entrada (caso os dados sejam do tipo analógico ou digital), também representado por um ator (por ser um elemento externo que interage diretamente com o Software Embarcado). Para este ator existe um caso de uso chamado "Recebe Dados de Entrada" cuja função é receber todos os dados provindos de todos os dispositivos. Os atores que enviam os dados diretamente ao Software, sem passar pelo caso de uso "Conversor de sinais de entrada" são do tipo serial.

Os casos de uso podem ser relacionados de três formas: generalização, inclusão e extensão. Essas relações potencializam significativamente a reutilização da especificação de requisitos, ou seja, elas permitem ao analista aquilo que o programador de linguagens orientadas a objetos normalmente pratica: reutilizar um trabalho já efetuado. Os relacionamentos entre os diagramas de casos de uso e os atores implementados neste trabalho são os seguintes: Inclusão (o qual está indicado pela palavra *include*) e os demais relacionamentos são do tipo generalização [5].

O caso de uso "Inicialização" é responsável pela comutação entre os casos de uso que executam as funções do Software Embarcado e entre o caso de uso das rotinas de interrupções (ISR), ele tem a opção de interromper a execução de qualquer tarefa do Software Embarcado, caso seja necessário, e inicializar (ou dar prosseguimento) a execução de outra. Os resultados processados pelo Software Embarcado são enviados aos atores de saídas.

Em aplicações espaciais, pode-se dividir o *software* em duas funções principais: Controle de Velocidade Angular (RCS) e Comando e Tratamento de Dados (C&DH) [7], os diagramas a seguir seguirão esta divisão.

3.3.2. Diagrama de Casos de uso do Controle de Velocidade Angular (RCS)

A Figura 3 apresenta o diagrama de caso de uso "Controle de Velocidade Angular", este diagrama é uma versão resumida do diagrama completo do Software Embarcado, contemplado apenas os atores e os casos de uso necessários para execução do cálculo da lei de controle e atuação dos propulsores.

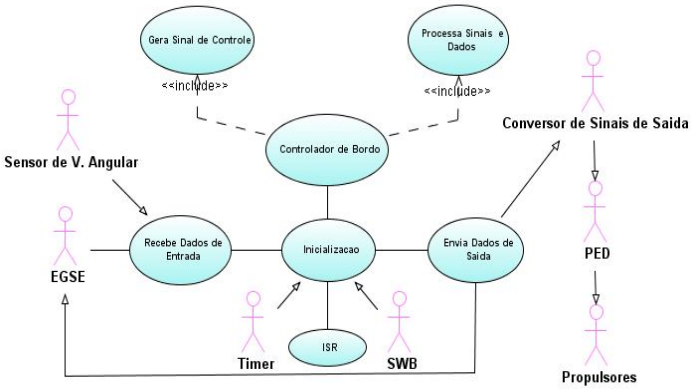

Figura 3. Diagrama de Caso de Uso Controle de Velocidade Angular (RCS)

3.3.3. Diagrama de Casos de uso de Comando e Tratamento de Dados (C&DH)

A Figura 4 exibe o diagrama de caso de uso "Comando e Tratamento de Dados", este diagrama é uma versão resumida do diagrama completo do Software Embarcado, contemplado os atores e os casos de uso necessários para o

processamento de sinais e dados, formação do pacote de telemetria no formato PCM (Dados de serviço e Dados de Carga Útil), comutação de energia, execução do Watchdog, processamento de telecomandos.

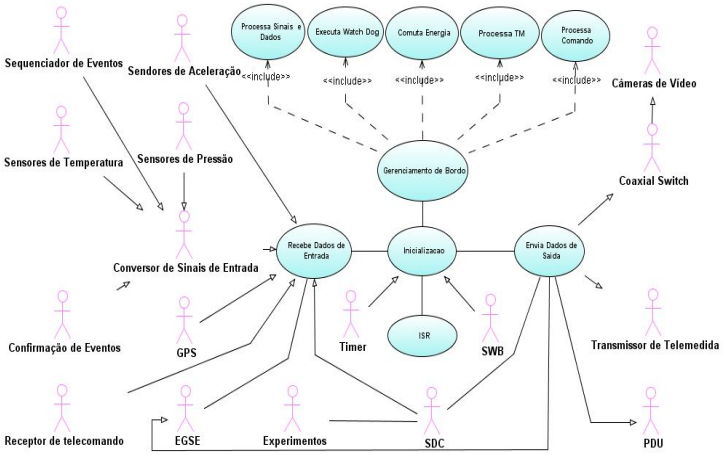

Figura 4. Diagrama de Caso de Uso "Comando e Tratamento de Dados (C&DH)"

3.3.4. Diagramas de Atividades

Definidos os casos de uso e atores do sistema, são então elaborados os diagramas de atividades, que detalham em alto nível as ações desempenhadas no cenário de cada caso de uso, mostrando cada passo como cada um deles é realizado.

O diagrama de atividades que representa o caso de uso "Inicialização" é apresentado na Figura 5 cuja função é inicializar o sistema, junto ao *Software* Básico (SWB), receber a freqüência do oscilador (devendo com isso cumprir o tempo a execução dos processos dentro de um ciclo), Solicitar as ISR e executar os casos de uso: Recebe Dados de Entrada, Controlador de Bordo, Gerenciamento de Bordo e Envia Dados de Saída.

O diagrama de atividades que representa o caso de uso "Gerenciamento de Bordo" é apresentado na Figura 6, cuja função é receber os dados provenientes dos atores, gerenciar e executar as rotinas para processar sinais e dados, telemetria, telecomando, comutação de energia, rotina de segurança (wathDog) através de funcionalidades fornecidas pelos casos de uso, e disponibiliza os resultados para o diagrama de caso de uso "Envia Dados de Saída". Caso a execução do *software* tenha sido finalizada, o diagrama passa para o estado final, em caso negativo, ele retorna para atividade, "*Wait*" e fica aguardando comando para se repetir o ciclo.

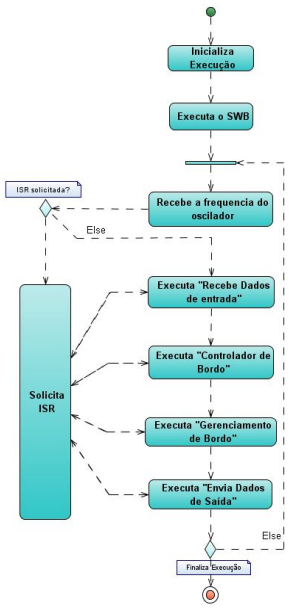

Figura 5. Diagrama de Atividade do caso de uso "Inicialização"

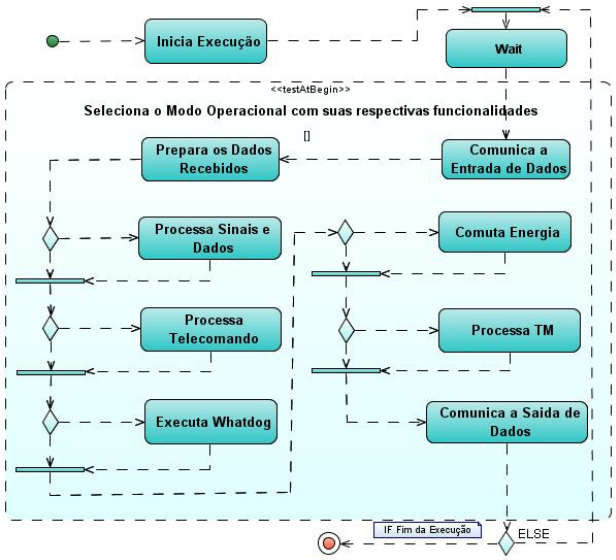

Figura 6. Diagrama de Atividade do caso de uso “Gerenciamento de Bordo”

O diagrama de atividades que representa o caso de uso “Controlador de Bordo” é apresentado na Figura 7, cuja função é receber os dados provenientes dos atores, gerenciar e calcular as leis de controle para atuação do veículo e disponibiliza os resultados para o diagrama de caso de uso “Envia Dados de Saída”. Caso a execução do *software* tenha sido finalizada, o diagrama passa para o estado final, em caso negativo, ele retorna para atividade, “Wait” e fica aguardando comando para se repetir o ciclo.

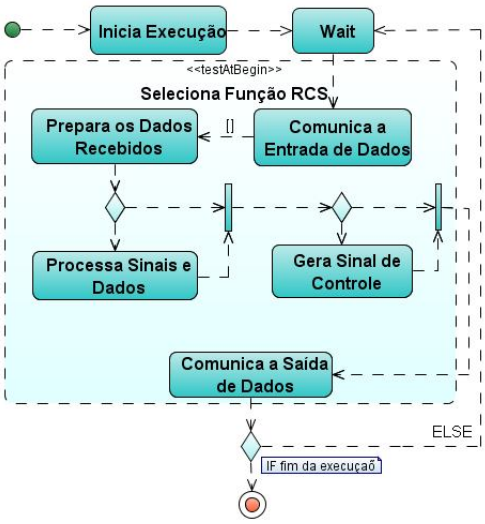

Figura 7. Diagrama de Atividade do caso de uso “Controlador de Bordo”

3.3.5. Diagramas de Seqüência

A função destes diagramas é apresentar em ordem cronológica de execução de tarefas entre os objetos.

O diagrama de seqüência que representa o caso de uso “Inicialização” é apresentado na Figura 8, que ilustra a execução do Software Embarcado, na seguinte ordem: receber a freqüência do oscilador, chamar as rotinas de interrupções ISR, execução do Software Básico. Posteriormente recebe os dados de entradas dos atores, executa o gerenciamento de bordo e finaliza enviando os dados de saídas para os atores.

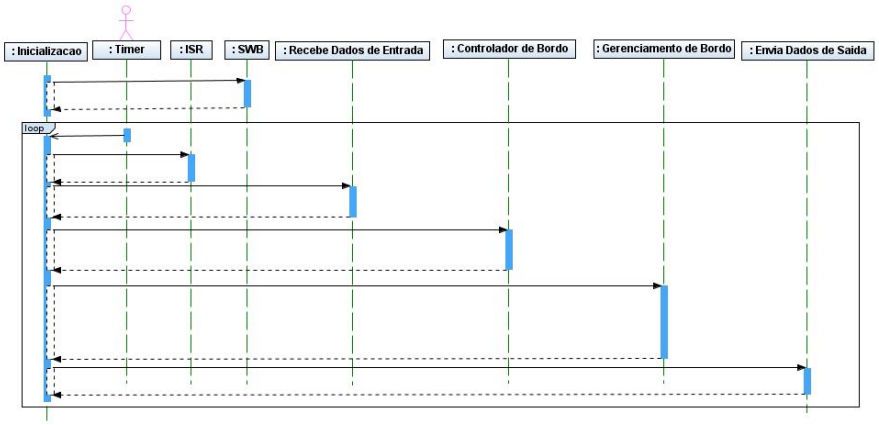

Figura 8. Diagrama de Seqüência do caso de uso “Inicialização”

O diagrama de seqüência que representa o caso de uso “Recebe Dados de Entrada” do C&DH é apresentado na Figura 9, este ilustra o cenário de recebimento dos dados de entrada provenientes dos atores com identificação “E” e “E/S” 2 da tabela 1. Todo o processo se encontra dentro de um loop que se repete na freqüência definida no oscilador até o fim da missão.

O diagrama de seqüência que representa o caso de uso “Gerenciamento de Bordo” do C&DH é apresentado na Figura 9, que após receber os dados do caso de uso “Recebe Dados de Entrada” os envia para os casos de uso responsáveis e executas as seguintes funções: processar sinais e dados, executar o Watchdog, gerenciamento de energia, executar o *HouseKeeping*, processar os comandos, executar funções de suporte, comutar sinais de vídeo, selecionar as faixas dos acelerômetros e processar as telemetrias, ao finalizar disponibiliza as informações para o caso de uso responsável pela saída de dados. Todo o processo se encontra dentro de um loop que se repete na freqüência definida no oscilador até o fim da missão.

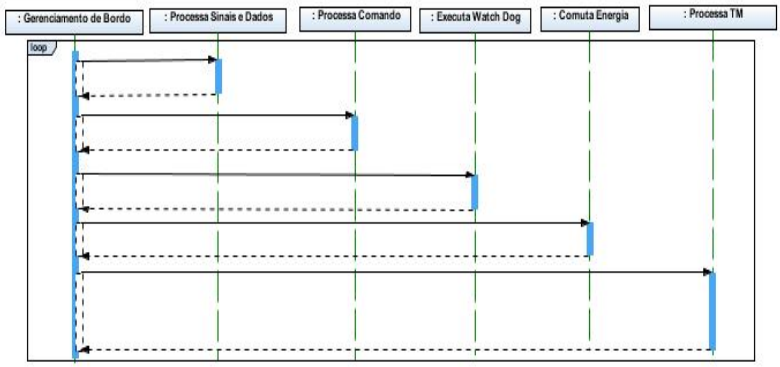

Figura 9. Diagrama de Seqüência do Caso de Uso “Gerenciamento de Bordo” do C&DH

O diagrama de seqüência que representa o caso de uso “Controlador de Bordo” do RCS é apresentado na Figura 10, que após receber os dados do caso de uso “Recebe Dados de Entrada” os envia para o caso de uso responsável por calcular as leis de controle e pelo caso de uso responsável por gerenciar e formatar os dados, ao finalizar disponibiliza as informações para o caso de uso responsável pela saída de dados. Todo o processo se encontra dentro de um loop que se repete na freqüência definida no oscilador até o fim da missão.

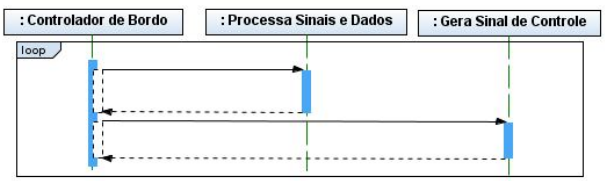

Figura 10. Diagrama de Seqüência do caso de uso “Controlador de Bordo” do RCS

3.3.6. Diagrama de Estados

Um diagrama de estados, também conhecido por diagrama de transição de estado ou por máquina de estados, permite modelar o comportamento interno de um determinado objeto, subsistema o sistema global [5]. Ele mostra os eventos que causam uma transição de um estado para o outro e ações que resultam de uma mudança de estado, devendo ser elaborado para classes de objetos que possuem comportamento dinâmico, pois para que o sistema passe de um estado a outro, é necessário que ocorra algum evento que constitua um gatilho (*trigger*) para esta mudança. As tarefas podem ser desempenhadas tanto na passagem de um estado a outro, quanto dentro do estado (em sua entrada ou em sua saída). Um estado pode possuir uma transição para ele mesmo.

A Figura 11 apresenta o diagrama de Estados "Acionamento dos Modos de Operação". Este diagrama inicia-se e entra em modo de espera no estado "*wait*", aguardando uma confirmação para executar, que se repete a cada ciclo programado pelo oscilador. Após iniciar um novo ciclo, o diagrama sai do estado "*wait*" e passa por uma série de teste IF, para definir em que modo operacional o ciclo será executado. mostrados a seguir:

1. Se o teste atender as exigências do "IF modo Propulsão" o diagrama de estados passa do estado "*wait*" para o estado "C&DH", note que para controle do pacote *Housekeeping* ainda há um segundo teste "IF antes do *Lift-off*".
2. Se o teste atender as exigências do "IF modo Controle" o diagrama de estados passa do estado "wait" para os estados "RCS" e "C&DH" respectivamente.
3. Se o teste atender as exigências do "IF modo uG" o diagrama de estados passa do estado "*wait*" para os estados "RCS" e "C&DH" respectivamente. Este teste booleano podes ser contemplado tanto antes do teste "IF modo controle" como depois, podendo o *software* entrar neste modo de operação mesmo sem antes ter passado pelo modo Controle.
4. Se o teste atender as exigências do "IF modo Flat-spin" o diagrama de estados passa do estado "*wait*" para os estados "RCS" e "C&DH" respectivamente.
5. Se o teste atender as exigências do "IF modo Recuperação" o diagrama de estados passa do estado "*wait*" para o estado C&DH.

Após executar este(s) estado(s) o diagrama de estados retorna para o estado "*wait*" e repete todo o processo até que a condição de fim da missão seja contemplada.

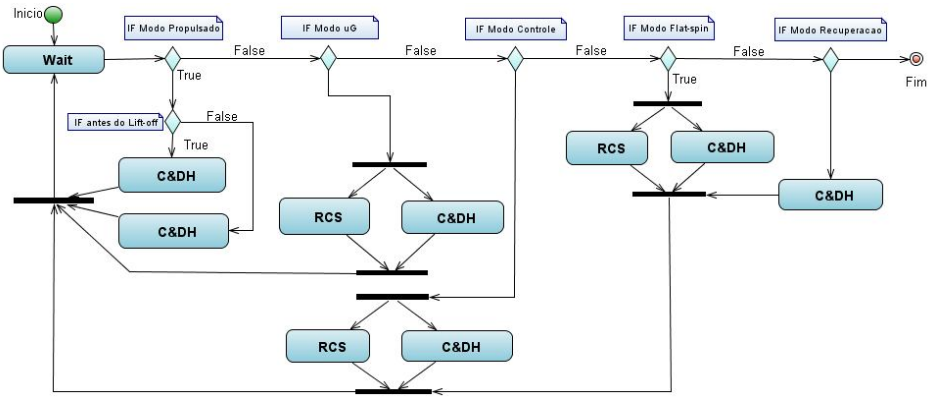

Figura 11. Diagrama de Estados "Acionamento dos Modos de Operação"

3.4. Codificação do Software

Com base nestes conceitos, a codificação deste sistema embarcado será desenvolvida utilizando a metodologia de programação orientada a objeto em C (OO em C) [12] [13], podendo fazer uso de todos os recursos que UML 2.0 oferece para V & V para *software*s mesmo antes da fase de codificação, e tendo flexibilidade de ser embarcado na maioria das plataformas hoje existentes, já que sua programação será em C.

4. CONCLUSÕES

A UML é uma linguagem unificada que auxilia na modelagem de sistemas de informações desenvolvidos no paradigma orientado a objetos (OO) [5]. Como o desenvolvimento de sistemas embarcados de tempo real utilizando linguagem orientada a objetos tem evoluído muito lentamente, principalmente para a análise, concepção e sua programação, a introdução de tecnologias orientadas a objeto, em alguns sistemas não acontece devido à falta de suporte C ++ para algumas plataformas, este artigo trás uma solução de desenvolvimento de *software*, na qual o sistema

de tempo real é modelado, validado e verificado (V & V) por diagramas UML, codificado em C, utilizando técnicas de OO em C, e com uso de ferramentas como LabView, Matlab e Simulink, pode-se garantir que cada *clock* do *software* consiga contemplar a execução de todos seus requisitos impostos, sem extrapolar este tempo nem terminado a execução antes muito menos depois, garantindo com isso um Sistema de Tempo Real Crítico (*hard* RTS).

6. AGRADECIMENTOS

Agradecemos a Orbital Engenharia LTDA pela oportunidade e apoio a esta publicação e a FINEP pelo financiamento do projeto.

7. REFERÊNCIAS

[1] J. J. Labrosse. Embedded Systems Building Blocks: Complete and Ready-to-Use Modules in C. 2º ed. San Francisco, CA: CMP, 2002. 611 p.
[2] P. C. Véras. Modelagem e análise do *software* embarcado de piloto automatic de um VANT. 2007. 135f. Tese de mestrado – ITA, São José dos Campos.
[3] ECSS-E-40 C Draft 10.0. Space Engineering – Software. European Cooperation for Space Standardization (ECSS), 15-03-2008.
[4] I. Sommerville. Engenharia de Software: Person – Prentice Hall. 8ª Edição, 2007. 552 p.
[5] R. A. Ramos. Treinamento Prático em UML: Desenvolva e Gerencie seus projetos com essa sensacional ferramneta. Universo dos livros editora Ltda. 1ª Edição, 2006. 144 p.
[6] A. C. Varoto. Visões em arquitetura de *software*. 2002. 99f. Dissertação de mestrado – IME, Rio de Janeiro.
[7] P. A. Laplante. Real-Time Systems Design and Anaysis: IEEE Press Editorial Board. 3ª Edition, 2004. 505 p.
[8] P. B. Douglas. Real-time UML developing efficient objects for embedded systems, 2ª edition. 2002.
[9] J. R. Wertz; W. J. Larson. Apace Mission Analysis and Design. Third edition. Space Technology Library; Californi, 1999. 969 p.
[10] L. F. E. Cocian. Manual da Linguagem C. ULBRA, 1º edição. 2004 496 p.
[11] A. BURNS; A. Wellings. Real-time systems and programming languages, Addison-Wesley, Reading, MA, USA, 1996.
[12] Object Oriented Programming in C; www.eventhelix.com/…/object. Acessado em 26/05/2009
[13] Object-Oriented Programming with ANSI-C http://www.planetpdf.com/codecuts/pdfs/ooc.pdf. Acessado em 26/05/2009.

7. NOTA DE RESPONSABILIDADE

Os autores são os únicos responsáveis pelo material incluído neste artigo.