Software Architecture for Redundant Computing Platform Embedded in Space Vehicles

Luciana Burgareli, Nanci Arai, Rovedy Busquim, Martha Abdala, Selma Melnikoff and Mauricio Ferreira

***Abstract*—Embedded software in space systems is critical and requires a well-defined and documented development process in their long life cycle. In this case, the software is part of a larger system that also includes the hardware that the software interacts with. Thus, many of the characteristics that must be considered in the software specification and design are directly related to the hardware components. The system architecture is a formal description of its building blocks, their properties and the interaction between them and is used to analyze characteristics, such as memory consumption, response time, performance, reliability, and safety. From both software and hardware most basic elements such as components and connectors to more complex properties such as behavior, an Architecture Description Language (ADL) is used in order to obtain a more accurate and precise description of the system architecture. This is accomplished by modeling the case study, a critical space software architecture, into a redundant embedded computational platform and analyzing it through the Architecture Analysis and Design Language (AADL). This work contributes to demonstrate that through fault analyses, AADL models can help to predict if restrictions, such as safety, will be fulfilled before the system construction.**

***Index Terms*—Embedded software, software architecture, software system.**

I. Introdução

Organizações militares envolvidas com o setor aeroespacial requerem o desenvolvimento de softwares específicos para atender a necessidades únicas, o qual deve seguir um ciclo de vida de desenvolvimento de software apropriado para garantir a qualidade do produto final [1]. Além disso, o desenvolvimento desse tipo de software apresenta inúmeros desafios, dentre eles, projetar uma arquitetura robusta, livre de falhas e com decisões de projeto mais adequadas [2].

A prática de Engenharia de Sistemas Baseada em Modelos (*Model-Based Systems Engineering* - MBSE) inclui a análise sistemática de modelos e arquitetura de software desde as fases iniciais do ciclo de vida. Para apoiar essa prática, são recomendadas modelagens formais e ferramentas de análises e desenvolvimento que operam na especificação arquitetural, como a AADL. A AADL é uma linguagem textual e gráfica direcionada para projeto e análise de arquiteturas de software e hardware de sistemas embarcados de tempo real [3] [4] e é caracterizada pelas propriedades que uma ADL deve prover como composição, abstração, reusabilidade, configuração, heterogeneidade e análise.

Este artigo propõe uma arquitetura de software em AADL, modelada na Open Source AADL Tool Environment (OSATE) [5], para uma plataforma computacional redundante embarcada no Veículo Lançador de Satélites (VLS) [6]. O VLS, controlado por um software embarcado de tempo real, foi desenvolvido pelo Instituto de Aeronáutica e Espaço (IAE), que é uma organização subordinada ao Departamento de Ciência e Tecnologia Aeroespacial (DCTA) da Força Aérea Brasileira.

A contribuição é demonstrar a possibilidade de se corrigir problemas nas fases anteriores a implementação do software, ao se incorporar a linguagem de modelagem ao processo de projeto de arquitetura e ao realizar a Análise de Impacto de Falhas (*Fault Impact Analysis* - FIA) e a Análise de Árvore de Falhas (*Fault Tree Analysis* - FTA).

Na seção II deste artigo, a AADL é brevemente introduzida. A seção III apresenta alguns trabalhos relacionados. Na seção IV, o experimento é descrito. Na seção V, os resultados são apresentados. Na seção VI, a conclusão é apresentada.

II. AADL

A AADL é uma linguagem para modelar e analisar sistemas constituídos de software, hardware, suas interações e propriedades. A AADL permite descrever as tarefas e a arquitetura de comunicação de um software embarcado, sua implantação/implementação em uma plataforma de hardware e sua interação com o sistema físico dentro de um modelo de arquitetura único e consistente [7].

A AADL é projetada para ser extensível de modo a acomodar análises que a linguagem *core* não suporta, tornando-a adaptável, de acordo com as necessidades dos usuários [4] [8]. Exemplos de anexos da linguagem são o Anexo de Comportamento, o Anexo de Modelo de Erro (*Error Model Annex*, Version 2 - EMV2) e o Anexo ARINC653.

O EMV2 aprimora os modelos de arquitetura da AADL com informações de falha para caracterizar condições anômalas. A partir desses modelos, podem-se realizar análises automatizadas de segurança e confiabilidade como a Análise de Modos e Efeitos de Falha (*Failure Modes and Effects Analysis* - FMEA)

L. A. Burgareli, N. N. Arai e R. A. B. e Silva são da Divisão de Aerodinâmica, Controle e Estruturas, Instituto de Aeronáutica e Espaço, São José dos Campos, SP 12228-904, Brasil (e-mail: lucianalab1@fab.mil.br, nancinna@fab.mil.br, rovedyrabs@fab.mil.br).

M. A. D. Abdala é tecnologista aposentada do Instituto de Aeronáutica e Espaço, São José dos Campos, SP 12228-904, Brasil (e-mail: martha.abdala@gmail.com).

S. S. S. Melnikoff é professora Titular da Escola Politécnica da Universidade de São Paulo, São Paulo, SP 05508-010, Brasil (e-mail: selma.melnikoff@usp.br).

M. G. V. Ferreira é pesquisador do Instituto Nacional de Pesquisas Espaciais, São José dos Campos, SP 12227-010, Brasil (e-mail: mauricio.ferreira@inpe.br).

e a FTA. Na OSATE, a FMEA é suportada pela FIA. O EMV2 permite especificar como um modo de falha pode restringir um conjunto de modos operacionais de um sistema modelado pela AADL *core* [9]. Esse mapeamento entre modos gera uma máquina de estados combinada e permite que a arquitetura do sistema seja modelada de forma mais completa.

III. Trabalhos Relacionados

As principais bases de publicações científicas pesquisadas foram: IEEE Xplore, ACM Digital Library e ScienceDirect da Elsevier.

Na área de sistemas críticos embarcados, a AADL foi empregada para modelar a arquitetura do sistema de voo da Missão Juno com intuito de analisar a geração de dados e memória e realizar análise de latência e fluxo de dados de ponta a ponta [10]. Essa última análise também foi utilizada em [11], porém realizada com a OSATE, para auxiliar os desenvolvedores a analisar e verificar desempenho dos produtos em fases anteriores ao desenvolvimento, detectar problemas potenciais no projeto, assim como reajustar o projeto aos requisitos de sistema. A diferença é que o primeiro trabalho utilizou o EMV2 para cobertura de falhas e, consequentemente, para descobrir propagações ausentes.

Ainda em sistemas críticos, a AADL foi usada para modelar a arquitetura de um sistema ferroviário para verificar e medir os *trades-offs* entre escalonamento e tempo de resposta de requisitos de disponibilidade e confiabilidade [12]. Diferentemente, em [13], além de apresentar um *background* sobre AADL em sistemas embarcados, o trabalho executou análises de falhas em um cenário de controle e coordenação de múltiplos sistemas subaquáticos com o EMV2. Utilizou a OSATE para realizar análise de latência e análise de risco.

Uma biblioteca de componentes da AADL, para facilitar a abordagem de modelagem de arquitetura e permitir a análise da arquitetura precoce de um sistema robótico com a OSATE, foi apresentada em [14]. De forma similar, em [15], foi descrita uma extensão da AADL, como um novo anexo de segurança, para suportar a modelagem do sistema sob condições de falhas e foi aplicada como estudo de caso em um sistema espacial.

Diferentemente dos trabalhos supracitados, este artigo modelou uma arquitetura redundante de um sistema crítico embarcado em AADL. Para complementar essa arquitetura e obter uma configuração dinâmica, foram definidos, com auxílio da OSATE, os modos operacionais que permitiram realizar as análises de falhas em conjunto com o EMV2. A FIA e a FTA indicaram os fluxos de erros e apresentaram algumas causas de possíveis erros. Além disso, o trabalho apresentou uma sequência de atividades planejada para modelagem e análise de falhas que pode ser incluída em um processo de arquitetura de software.

A TABLE I apresenta o comparativo entre este artigo e os trabalhos relacionados. A primeira coluna contém a identificação dos artigos (ID) e as demais se referem aos critérios de comparação: se o artigo pertence a área de sistemas críticos, embarcados ou de tempo real (CET); se utiliza OSATE (OSA); se realiza Análise de Falhas (AF); se emprega o EMV2 (EMV2); se apresenta a FTA (FTA) e a FIA (FIA).

TABELA I
Comparativo entre os trabalhos relacionados.

ID	CET	OSA	AF	EMV2	FTA	FIA
[10]	√		√	√		
[11]	√	√				
[12]	√					
[13]	√	√	√	√		
[14]	√	√				
[15]	√		√			
Este artigo	√	√	√	√	√	√

IV. Experimento

Para realizar o experimento, são executadas as atividades da abordagem proposta na Fig. 1 com o suporte da OSATE, versão 2.10.2. A primeira atividade é a definição do modelo da AADL *core*, baseada nos documentos de especificação de requisitos e de projeto de software. Nessa fase, para definir o modelo da arquitetura estática redundante, utilizou-se a técnica de redundância *Hot Standby*. A segunda atividade é a inclusão de modos operacionais do sistema, o que resulta no modelo da arquitetura dinâmica redundante. A próxima atividade é a incorporação do EMV2, gerando o modelo de erro para viabilizar a avaliação de propriedades de segurança e confiabilidade do sistema. A realização do mapeamento entre modos operacionais e estados de comportamento de erro é necessária para definir o comportamento final do modelo. A execução da FIA e FTA deve ser realizada na atividade seguinte.

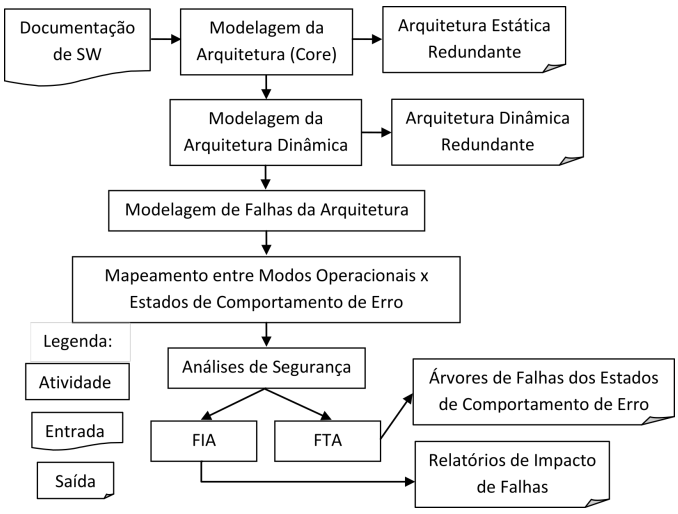


Fig. 1. Atividades da abordagem proposta.

A abordagem foi aplicada ao estudo de caso do software embarcado de tempo real do VLS, que permite, entre outras funções, a execução das decisões de controle e sequência de eventos do VLS. A realização da guiagem, navegação e o controle de atitude do VLS desde a sua decolagem é função da sua Rede Elétrica de Controle (REC). Em solo, as atividades das fases de testes e preparação para lançamento são realizadas pelo Banco de Controle (BC).

A. Modelagem da Arquitetura (Core)

Primeiramente, a modelagem da arquitetura do estudo de caso foi realizada somente com a especificação da AADL *core*, ou seja, sem os anexos da linguagem. Foram adotados os termos subcomponente, sistema e componente. Cada elemento de um modelo da AADL é referenciado como um subcomponente (*component* e/ou *subcomponent*), o qual pode ser um sistema (*system*) ou um componente (*device*).

Como um dos focos do trabalho é a arquitetura redundante e sendo o software do VLS legado e complexo, para utilizá-lo como estudo de caso, optou-se por uma versão adaptada na qual foram inseridos, aos modelos da AADL, novos sistemas e componentes e desconsiderados e/ou simplificados outros que não impactariam na modelagem da arquitetura redundante.

A arquitetura do estudo de caso é composta pelo sistema do BC (SisBC) e da REC (SisREC), ambos interligados pelo barramento Serial 422 (Ba422). O sensor do SisREC é representado pelo componente MSI, que envia os dados a dois controladores digitais redundantes, denotados pelos sistemas CD1 e CD2, para o processamento desses dados e os resultados de ambos os sistemas são enviados ao sistema AC que opera como um gerenciador decidindo qual resultado passa aos módulos de comando do sistema de tubeira móvel, representado pelo componente CTM. O SisREC é ainda interligado a outros sistemas do VLS, não modelados neste trabalho, através do barramento MIL-STD-1553 (Ba_1553).

O CD2 foi adicionado para atuar como reserva do módulo principal CD1. Na técnica redundante selecionada, a *Hot Standby*, quando o módulo inicial (CD1) apresentar falha, o módulo em espera (CD2) entra em operação e o AC passa a enviar os dados recebidos do CD2 ao CTM. Dessa forma, obtém-se o modelo da arquitetura estática redundante do software do VLS, conforme ilustrado na Fig. 2.

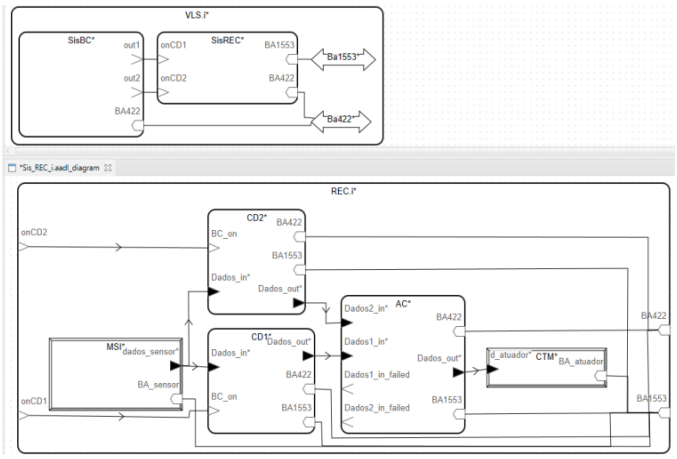


Fig. 2. Modelo da arquitetura estática redundante.

B. Modelagem da Arquitetura Dinâmica

Para complementar o modelo da arquitetura estática redundante do software do VLS, uma configuração dinâmica foi adicionada através dos modos operacionais que permitem modelar os vários estados operacionais dos subcomponentes [8]. Na OSATE, os modos operacionais representam uma configuração dinâmica dos subcomponentes que compõem o modelo, tornando-os ativos ou não, dependendo da ocorrência das transições dos modos.

Para o SisREC seguir a técnica *Hot Standby*, foram definidas as seguintes configurações:

1) Inicial: seus subcomponentes estão ativos, exceto o CD1 e o CD2 que são ativados pelo SisBC através das portas de entradas de eventos onCd1 e onCd2;
2) Operacional Redundante: o CD1 e o CD2 estão ativos, ou seja, devem receber dados do MSI e enviar para o AC. Entretanto, assume-se que o AC apenas envie ao CTM, os valores recebidos do CD1. Todos os outros subcomponentes também estão ativos;
3) Operacional Não Redundante: podem ocorrer duas situações: (1) o CD2 falha e o CD1 se mantém ativo e o AC envia dados do CD1 ao CTM; (2) e o CD1 falha e o CD2 se mantém ativo, e passa a ser o módulo principal do *Hot Standby*, fazendo com que o AC passe a enviar dados do CD2 ao CTM. Todos os outros subcomponentes estão ativos;
4) Sistema Falho: o CD1 e o CD2 falham, ou pelo menos um dos outros subcomponentes falha provocando a falha no SisREC.

Na especificação da AADL, no SisREC, essas configurações são representadas através dos modos operacionais:

1) Initialize: configuração Inicial;
2) Red: configuração Operacional Redundante;
3) CD1_OK: configuração Operacional Não Redundante na situação 1;
4) CD2_OK: configuração Operacional Não Redundante na situação 2;
5) Stop: configuração Sistema Falho.

Dessa forma, obtém-se o modelo da arquitetura dinâmica resultante da SisREC, com o acréscimo do diagrama de estados, que representa os modos e como as transições são disparadas a partir dos eventos que partem das portas de eventos dos subcomponentes, conforme apresentado na Fig. 3. Essa figura também ilustra o modo CD2_OK selecionado, na qual os módulos ativados pela OSATE podem ser observados destacados na cor rosa.

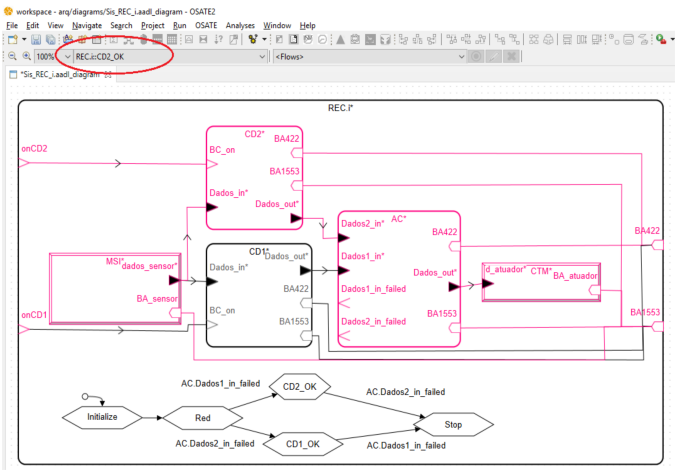


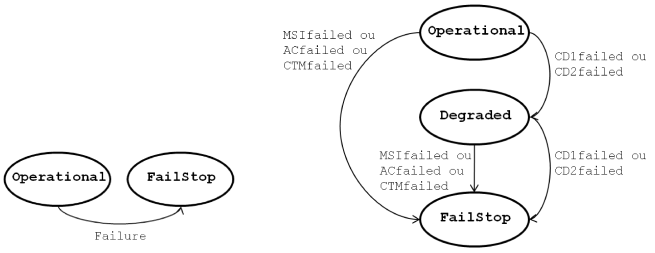
Fig. 3. Modelo da arquitetura dinâmica redundante.

C. Modelagem de Falhas da Arquitetura

Considerando-se os modelos da AADL gerados nas seções IV-A e B, foi acrescentado o anexo de erro nos tipos e/ou implementações dos subcomponentes. O anexo de erro apresenta anotações específicas a cada subcomponente e apropriadas para a análise que se pretende executar. A seguir são apresentadas as bibliotecas e as anotações de erros utilizadas.

1) Biblioteca de Modelos de Erro: Foi utilizada a biblioteca de modelos de erro *ErrorLibrary* que contém várias definições de tipos de erro e especificações reusáveis de máquinas de estado de comportamento de erro, dentre as quais foi selecionada a *FailStop*, apresentada na Fig. 4a, que é usada pelo MSI, CD1, CD2, AC e CTM.

Além disso, foi definida uma biblioteca de modelos de erro específica, denominada VLSLibrary, na qual foi inserida a máquina de estados de comportamento de erro DegradedREC, ilustrada na Fig. 4b, que é usada pelo SisREC.



(a) *FailStop* (b) DegradedREC

Fig. 4. Máquinas de estado de comportamento de erro.

2) Propagação de erros: A seção *error propagations* declara como um erro pode ser propagado pela arquitetura. No caso do AC, como pode ser visualizada nas linhas 12-22 da Fig. 5, foram declaradas, como pontos de propagação, duas portas de entrada que recebem um erro propagado do CD1 e do CD2 e uma porta de saída que propaga um erro para o CTM. Os fluxos de erro detalham quando o subcomponente é uma fonte de erro (*error source*) ou um caminho de erro (*error path*) ou um término de erro (*error sink*). Essas condições, sob as quais o subcomponente propaga ou mascara o erro de uma entrada de erro, são detalhadas na especificação do *component error behavior*.

3) Comportamento de erro de componente: A seção *component error behavior* representa o comportamento anômalo de um subcomponente da AADL através de uma máquina de estados e pode representar falhas dos subcomponentes, gerenciamento e propagação de erros. No caso do SisREC, como pode ser visualizada nas linhas 26-43 da Fig. 5, a máquina de estados DegradedREC foi detalhada através da declaração dos eventos de erro e das transições específicas desse sistema. Na seção *component error behavior* do SisREC, foram declarados os eventos de erro que podem ocorrer para esse sistema e as possíveis transições entre seus estados de erro, causadas por esses eventos.

4) Comportamento de erro composto: A seção *composite error behavior* permite definir o estado de comportamento de erro de um subcomponentes em termos dos estados de comportamento de erro de seus elementos. No caso do SisREC,

```
1.  system AC
2.  features
3.              Dados1_in: in data port;
4.              Dados2_in: in data port;
5.              Dados_out: out data port;
6.              Dados1_in_failed:out event port;
7.              Dados2_in_failed: out event port;
8.              BA422: requires bus access Hw::Serial422;
9.              BA1553: requires bus access Hw::MILSTD1553B;
10.   annex emv2{**
11.   use types ErrorLibrary;
12.   error propagations
13.             Dados1_in: in propagation {BadValue};
14.             Dados2_in: in propagation {BadValue};
15.             Dados_out: out propagation {BadValue};
16.   flows
17.             ACError: error source Dados_out{BadValue};
18.             CD1Error_path: error path Dados1_in ->
        Dados_out {BadValue};
19.             CD2Error_path: error path Dados2_in ->
        Dados_out {BadValue};
20.             CD1Error_sink: error sink Dados1_in {BadValue};
21.             CD2Error_sink: error sink Dados2_in {BadValue};
22.   end propagations;
23. **};
24. end AC;
25. ....
26.   component error behavior
27. events
28.             CD1failed: error event;
29.             CD2failed: error event;
30.             ACfailed: error event;
31.             MSIfailed: error event;
32.             CTMfailed: error event;
33. transitions
34.             Operational-[CD1failed or CD2failed]-> Degraded;
35.             Degraded -[ CD1failed or CD2failed]-> FailStop;
36.             Operational-[ACfailed]-> FailStop;
37.             Degraded -[ACfailed]-> FailStop;
38.             Degraded -[MSIfailed]-> FailStop;
39.             Degraded -[CTMfailed]-> FailStop;
40.             Operational -[ACfailed]-> FailStop;
41.             Operational -[MSIfailed]-> FailStop;
42.             Operational -[CTMfailed]-> FailStop;
43. end  component;
```

Fig. 5. Implementação do modelo de erro na AADL.

como pode ser visualizada nas linhas 5-15 da Fig. 6, quando os seus subcomponentes MSI, CD1, CD2, AC e CTM estão operacionais, o SisREC também está operacional. Quando ou o CD1 ou o CD2 estiver em falha, a SisREC opera em estado degradado. Finalmente, se ambos o CD1 e o CD2 falharem ou se um dos demais subcomponentes falhar, a SisREC estará em falha.

D. Mapeamento entre Modos Operacionais e Estados de Comportamento de Erro

Os estados de comportamento de erro do EMV2 são modos de falha (*failure*) e podem especificar restrições aos modos operacionais do modelo da AADL *core* de um subcomponente, estabelecendo quais modos operacionais podem estar ativos e quais transições de modo podem ser iniciadas quando o subcomponente está em um estado de comportamento de erro específico. No mapeamento de modo, a função de um estado de comportamento de erro é especificar um conjunto de restrições nos modos impostos por estados de comportamento

de erro. A especificação do mapeamento sobrepõe os estados de comportamento de erro da máquina de estados e modos (*mode state machine*).

O SisREC tem os modos operacionais e seus estados de comportamento de erro ilustrados, respectivamente, na Fig. 3 e na Fig. 4b. A partir do mapeamento dos estados de comportamento de erro aos modos, é possível derivar uma máquina de estado de comportamento combinada, conforme ilustrado na Fig. 7 e, assim, definir de forma mais completa, o comportamento final do modelo. A implementação desse mapeamento pode ser visualizada nas linhas 18-21 da Fig. 6.

E. Análises de Segurança

Os resultados da FTA e da FIA e parte da especificação textual do anexo de erro dos subcomponentes são apresentados nas próximas subseções.

```
1. annex emv2{**
2. use types ErrorLibrary;
3. use behavior VLSLibrary::DegradedREC;
4.
5. composite error behavior
6. states
7.   [MSI.Operational and CD1.Operational and CD2.Operational
8.   and AC.Operational and CTM.Operational]->Operational;
9.   [MSI.FailStop]->FailStop;
10.  [CD1.FailStop and CD2.FailStop]->FailStop;
11.  [AC.FailStop]->FailStop;
12.  [CTM.FailStop]->FailStop;
13.  [MSI.Operational and AC.Operational and CTM.Operational
14.  and (CD1.FailStop or CD2.FailStop)]->Degraded;
15. end composite;
16.
17. component error behavior
18. mode mappings
19.  Operational in modes (Initialize, Red);
20.  Degraded in modes (CD1_OK,CD2_OK);
21.  FailStop in modes (Stop);
22. end component;
23. **};
```

Fig. 6. Comportamento de erro composto e mapeamento entre modos operacionais e estados.

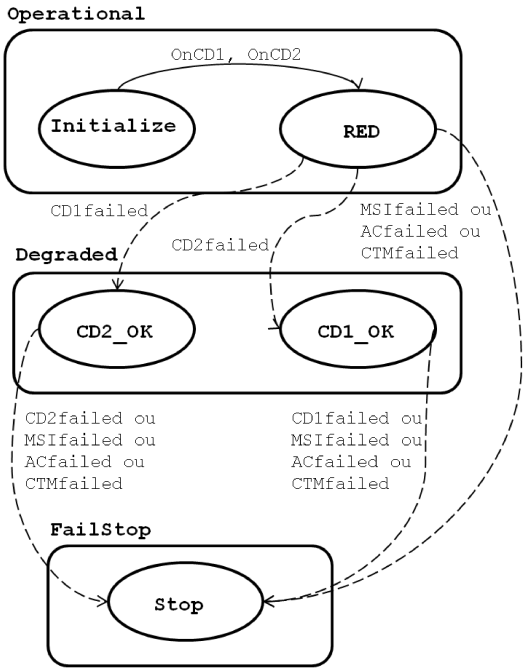


Fig. 7. Máquina de estado de comportamento combinada.

1) FIA: Seu objetivo é rastrear os fluxos de erro entre a fonte de erro e o subcomponente afetado, mostrando o impacto de falhas pela arquitetura do sistema. A saída é um relatório no formato de tabela.

A FIA pode ser realizada em um processo incremental, sendo executada em modelos da AADL anotados com EMV2 em vários níveis de detalhes. Minimamente, os modelos devem ter especificadas propagações de erro de entrada e saída. Entretanto, para que a FIA forneça um resultado mais fidedigno, pode ser adicionado ao modelo o *component error behavior* com seus eventos de erro, estados de erro e transições. Dessa forma, foram realizados testes em dois cenários:

- Cenário 1: versão simplificada de um anexo de erro ao modelo da AADL *core* do estudo de caso. Essa versão possui apenas especificações de propagação de fluxo de erros de entrada e saída (fontes de erro, caminhos de erro e términos de erro), que são as especificações mínimas que a análise FIA precisa para ser realizada;
- Cenário 2: o anexo de erro do Cenário 1 aprimorado. Essa versão possui máquina de estados definida para o SisREC (DegradedREC), bem como o *component error behavior* especificado com suas transições e propagações de erros e o *composite error behavior* encerrada através do *error sink*. Os subcomponentes CD1, CD2, AC e CTM são impactados pela ocorrência de uma falha no MSI.

Na Tabela II apresenta-se o relatório da FIA para o SisREC considerando o Cenário 1. Por exemplo, nesse relatório, pode ser constatado que o impacto de falhas partindo do MSI ocorre em quatro linhas:

- Primeira linha MSI: Passa para CD1, vai para AC e é mascarado;
- Segunda linha MSI: Passa para CD1, vai para AC, vai para CTM e é mascarado;
- Terceira linha MSI: Passa para CD2, vai para AC, vai para CTM e é mascarado;
- Quarta linha MSI: Passa para CD2, vai para AC e é mascarado.

Quando o MSI se encontra em falha, todos os subcomponentes do SisREC são afetados pelo erro no sensor. O MSI pode falhar e propagar esta falha para o CD1 e CD2, os quais, por sua vez, também podem falhar e propagar as falhas do MSI para o AC. O AC também pode falhar e propagar ou parar as falhas do CD1 e CD2. Finalmente, se o CTM receber a falha do AC, a falha é tratada através do *error sink*.

2) FTA: A OSATE permite visualizar uma árvore de falhas criada automaticamente a partir de um modelo da AADL anotado com especificações do EMV2, através da instância da implementação do subcomponente com a condição de falha que se tornará o evento primário/raiz (*root*) da árvore. A ferramenta suporta a criação de árvores a partir de especificações de comportamento de erro composto e baseada em fluxos.

No anexo de erro do SisREC, o *composite error behavior* foi definido para os estados *Operational*, *Degraded* e *FailStop* em função dos estados de erro de seus subcomponentes. O estado *Operational* ocorre apenas quando todos os subcomponentes estão no estado *Operational*. O estado *Degraded* ocorre

TABELA II
RELATÓRIO DO IMPACTO DE FALHAS PARA O MSI – CENÁRIO 1

Component 1	Initial Failure Mode	1st Level Effect 3	Failure Mode	Second Level Effect	Failure Mode	Third Level Effect	Failure Mode
SisREC.CD2	AboveRange	AboveRange Dados_out → SisREC.AC:Dados2_in	SisREC.AC AboveRange	AboveRange Dados_out → SisREC.CTM:d_atuado	SisREC.CTM AboveRange [Masked]		
SisREC.CD2	AboveRange	AboveRange Dados_out → SisREC.AC:Dados2_in	SisREC.AC AboveRange [Masked]				
SisREC.MSI	AboveRange	AboveRange dados_sensor → SisREC.CD1:Dados_in	SisREC.CD1 AboveRange	SisREC.AC AboveRange [Masked]			
SisREC.MSI	AboveRange	AboveRange dados_sensor → SisREC.CD1:Dados_in	SisREC.CD1 AboveRange	AboveRange Dados_out → SisREC.AC:Dados1_in	SisREC.AC AboveRange	AboveRange Dados_out → SisREC.CTM:d_atuador	SisREC.CTM AboveRange [Masked]
SisREC.MSI	AboveRange	AboveRange dados_sensor → SisREC.CD2:Dados_in	SisREC.CD2 AboveRange	AboveRange Dados_out → SisREC.AC:Dados2_in	SisREC.AC AboveRange	AboveRange Dados_out → SisREC.CTM:d_atuador	SisREC.CTM AboveRange [Masked]
SisREC.MSI	AboveRange	AboveRange dados_sensor → SisREC.CD2:Dados_in	SisREC.CD2 AboveRange	SisREC.AC AboveRange [Masked]			
SisREC.CD1	AboveRange	AboveRange Dados_out → SisREC.AC:Dados1_in	SisREC.AC AboveRange [Masked]				
SisREC.CD1	AboveRange	AboveRange Dados_out → SisREC.AC:Dados1_in	SisREC.AC AboveRange	AboveRange Dados_out → SisREC.CTM:d_atuador	SisREC.CTM AboveRange [Masked]		
SisREC.AC	AboveRange	AboveRange Dados_out → SisREC.CTM:d_atuador	SisREC.CTM AboveRange [Masked]				

quando ou o CD1 ou o CD2 estiver no estado *FailStop*. O estado *FailStop* ocorre apenas quando os dois subcomponentes redundantes, CD1 e CD2, ou um dos demais subcomponentes, MSI, AC e CTM, estiver no estado *FailStop*, como mostrado na Fig. 8.

Cada subcomponente da SisREC possui seu anexo de erro que especifica como o fluxo de erro se propaga entre os subcomponentes, tanto de hardware quanto de software. O Ba_1553 e Ba_422 propagam seus erros, através do *bindings*. O MSI pode ser uma fonte de erro e propagar esse erro pela porta de saída. O CTM pode receber um erro pela porta de entrada, mas o fluxo de erro encerra-se no próprio CTM com um *error sink*. O CD1 e o CD2 podem ser fonte de erro e receber e propagar erros (*error path*). O AC é responsável pela propagação ou término do erro, conforme o comportamento dos sistemas redundantes. Na especificação do tipo AC, foi definida a propagação de erro, na qual foram especificados os fluxos que podem ser uma fonte de erro, um caminho de erro ou um ponto de término do erro. Na implementação do AC foi definido o *component error behavior*, no qual foram detalhadas as propagações: no caso de o subcomponente estar no estado *FailStop* ou no estado *Operational* e chegar erros nas duas portas de entrada, o erro será propagado; se estiver no estado *Operational* e chegar um erro em apenas uma das portas de entrada, o erro não será propagado.

V. RESULTADOS

Como resultados da realização desse trabalho, observou-se que hardware e software do estudo de caso puderam ser documentados em um único modelo de arquitetura. Constatou-se que, o EMV2 foi definido especificamente para cada análise proposta, gerou várias simulações de falhas com níveis de detalhes diferentes. Através da FIA foi possível rastrear os fluxos de erro entre uma fonte de erro e um subcomponente afetado, mostrando a propagação de falhas pela arquitetura do sistema. Com o diagrama de FTA, foi possível determinar, a partir da ocorrência de uma falha, algumas de suas causas básicas. Além desse conjunto de resultados auxiliar a prever restrições nas fases anteriores ao desenvolvimento, pode orientar e incentivar novas aplicações da AADL e do EMV2 em sistemas críticos similares.

Como lições aprendidas, além da capacitação na AADL e na OSATE, também se obteve conhecimentos na área de dependabilidade através dos estudos de técnicas de redundância e de análise de falhas. No estudo de caso, percebeu-se que foi possível gerar a FIA apenas com os fluxos de erros. Porém, as tabelas revelaram mais detalhes, ao considerar a máquina de estados, devido à inclusão de seus erros e transições. Na FTA notou-se que, para um determinado subcomponente, a inclusão do *composite* juntamente com as transições, fez com que as transições fossem ignoradas e apenas as regras do *composite* fossem obedecidas. As Árvores de Falhas apresentaram apenas os erros internos e não os propagados e, dessa forma, as *properties* (propriedades que descrevem as folhas com erros dos subcomponentes) não funcionaram. Por outro lado, sem o *composite*, apenas as transições foram obedecidas e a Árvore de Falhas apresentou os erros propagados e, nesse caso, as *properties* funcionaram.

Uma dificuldade encontrada foi o fato de a OSATE apresentar atualizações durante o desenvolvimento deste artigo que disponibilizou versões novas, com eventuais *bugs* corrigidos e características adicionais, porém alterou e/ou excluiu recursos que estavam sendo utilizados, ocasionando adaptações no trabalho para se adequar a essas atualizações. Por exemplo, a representação gráfica do diagrama de Árvores de Falhas foi modificada e o diagrama de impacto de falhas, que existia em versões anteriores, foi excluído.

VI. CONCLUSÃO

A principal contribuição deste artigo é apresentar a experiência de utilização da AADL para modelagem de uma arquitetura redundante no estudo de caso que, devido à sua natureza, é considerado crítico, exigindo maior confiabilidade em seu funcionamento. Essa arquitetura redundante deve prover uma operação normal, mesmo na presença de falhas, visando maior disponibilidade e segurança do sistema. Dessa forma, esse trabalho apresenta uma sequência de atividades organizada para facilitar o uso da AADL e suas anotações de erros. Esse conjunto de atividades pode auxiliar as equipes de desenvolvimento de software a incorporar, de forma planejada, a utilização da linguagem de modelagem ao seu processo de projeto de arquitetura de software, aprimorando a confiabilidade dos modelos gerados e provendo a realização de análises de segurança nas fases anteriores ao desenvolvimento do software, evitando retrabalhos. Através do desenvolvimento deste trabalho, conclui-se que a AADL é uma linguagem de

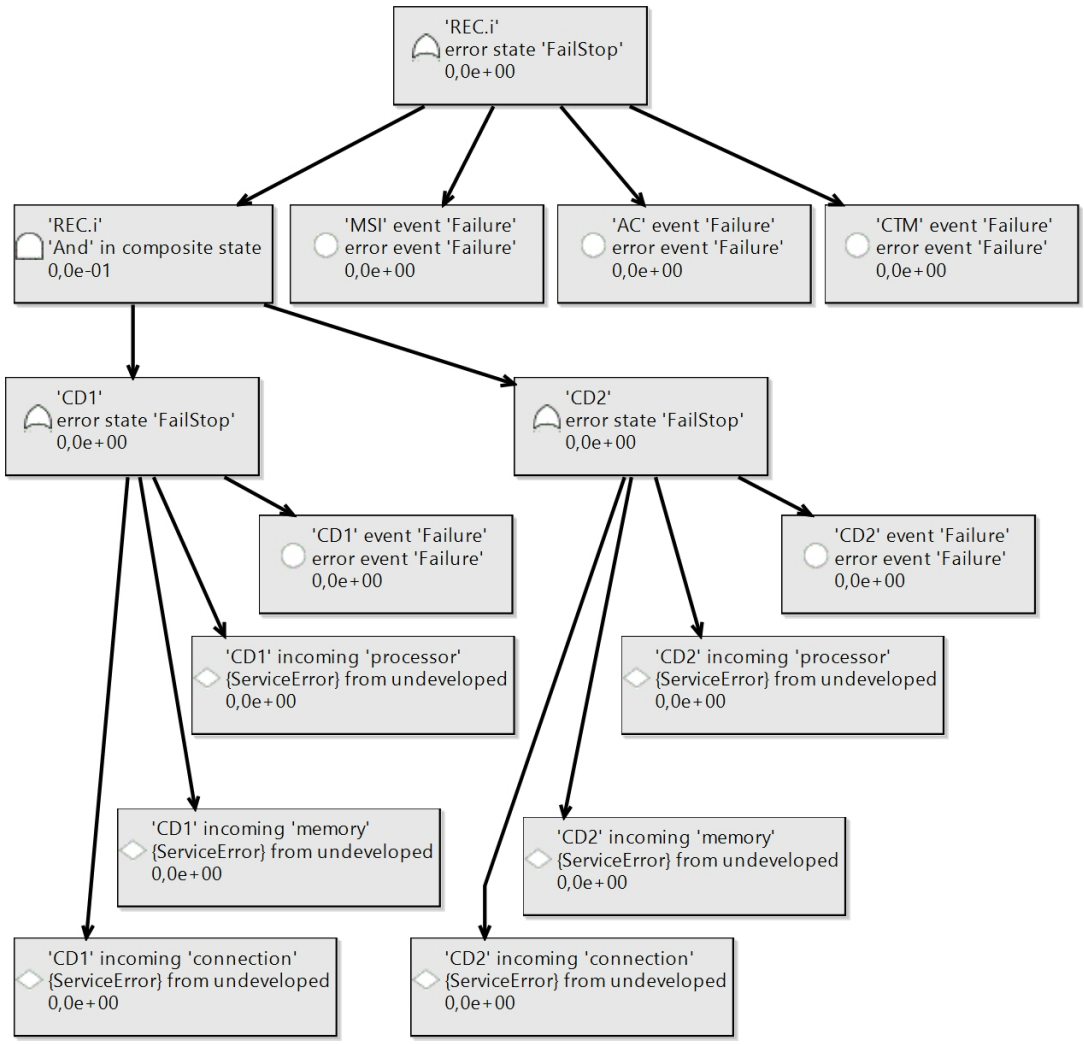


Fig. 8. Árvore de Falhas gerada a partir do modelo.

modelagem que fornece apoio para uma descrição completa da arquitetura, porque permite unir software e hardware em um único modelo incluindo suas interações e propriedades, diferentemente da arquitetura original do software do VLS, no qual os modelos de arquiteturas de hardware e software são projetados individualmente. Através das análises de falhas que a linguagem proporciona juntamente com a OSATE, a AADL fornece recursos para aumentar a segurança do modelo proposto. Como trabalho futuro, pretende-se aplicar a abordagem apresentada em outros softwares críticos a fim de se incorporar a utilização da AADL ao processo de desenvolvimento de software, juntamente com as análises de segurança de forma automatizada.

AGRADECIMENTOS

Nossos agradecimentos ao apoio da Agência Espacial Brasileira que financiou parte desse trabalho.

REFERÊNCIAS

[1] R. R. J. Jardim, M. Santos, E. Neto, E. da Silva, and F. de Barros, "Integration of the waterfall model with iso/iec/ieee 29148:2018 for the development of military defense system," *IEEE Latin America Transactions*, vol. 18, no. 12, pp. 2096–2103, 2020.

[2] S. Cook and G. Haverkamp, "Challenges and Opportunities for Software Development and Verification on Military Aircraft Systems," in *AIAA Scitech 2020 Forum*. Orlando, FL: American Institute of Aeronautics and Astronautics, Inc., 01 2020.

[3] Software Engineering Institute, "Architecture Analysis and Design Language (AADL)," https://www.sei.cmu.edu/our-work/projects/display.cfm?customel_datapageid_4050=191439, Carnegie Mellon University, feb 2022, acessado em 29/03/2023.

[4] J. Hudak and P. Feiler, "Developing AADL Models for Control Systems: A Practitioner's Guide," Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, Tech. Rep. CMU/SEI-2007-TR-014, 2007.

[5] Carnegie Mellon University, "Open Source AADL Tool Environment (OSATE)," https://osate.org/, acessado em 29/03/2023.

[6] Instituto de Aeronáutica e Espaço, "VLS-1," https://iae.dcta.mil.br/index.php/todos-os-projetos/todos-os-projetos-desenvolvidos/projetos-vls1, Departamento de Ciência e Tecnologia Aeroespacial, may 2019, acessado em 29/03/2023.

[7] P. Feiler and J. Delange, "Automated Fault Tree Analysis from AADL Models," *Ada Lett.*, vol. 36, no. 2, p. 39–46, may 2017.

[8] P. Feiler and D. Gluch, *Model-Based Engineering with AADL: An Introduction to the SAE Architecture Analysis & Design Language*. Addison-Wesley Professional, 2012.

[9] P. Feiler, J. Hudak, J. Delange, and D. Gluch, "Architecture Fault Modeling and Analysis with the Error Model Annex, Version 2,"

Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, Tech. Rep. CMU/SEI-2016-TR-009, 2016.
[10] M. Muñoz Fernández, "Using AADL to enable MBSE for NASA space mission operations," in *SpaceOps 2014 Conference*. Pasadena, CA: American Institute of Aeronautics and Astronautics, Inc, 2014.
[11] H. Yu and Y. Yang, "Latency Analysis of Automobile ABS Based on AADL," in *2012 International Conference on Industrial Control and Electronics Engineering*. IEEE, 2012, pp. 1835–1838.
[12] P. Crisafulli, D. Blouin, F. Caron, and C. Maxim, "Engineering Railway Systems with an Architecture-Centric Process Supported by AADL and ALISA: an Experience Report," in *10th European Congress on Embedded Real Time Software and Systems (ERTS 2020)*, Toulouse, France, Jan. 2020.
[13] H. A. PHAM, T. SORIANO, and V. H. NGO, "Applying AADL to realize embedded control systems for coordination of multiple low-cost underwater drones," in *OCEANS 2019 - Marseille*, 2019, pp. 1–7.
[14] E. Senn, L. W. J. Bourdon, and D. Blouin, "Multi-Paradigm Modeling for Early Analysis of ROS-Based Robotic Applications Using a Library of AADL Models," in *Proceedings of the 25th International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings*, ser. MODELS '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 677–683.
[15] D. Stewart, J. J. Liu, D. Cofer, M. Heimdahl, M. W. Whalen, and M. Peterson, "AADL-Based safety analysis using formal methods applied to aircraft digital systems," *Reliability Engineering & System Safety*, vol. 213, p. 107649, 2021.

Luciana Akemi Burgareli PhD in Electrical Engineering with emphasis on Digital Systems at Escola Politécnica de São Paulo, University of São Paulo – USP (2009). Senior Technologist at the Institute of Aeronautics and Space (IAE) of the Department of Aerospace Science and Technology (DCTA). Experience in software engineering.

Nanci Naomi Arai Master in Applied Computing by the National Institute for Space Research – INPE (2001). Research Assistant at the Institute of Aeronautics and Space (IAE) of the Department of Aerospace Science and Technology (DCTA). Experience in software engineering and real-time systems development.

Rovedy Aparecida Busquim e Silva PhD in Electronic and Computer Engineering from the Aeronautics Institute of Technology – ITA (2013). Senior Technologist at the Institute of Aeronautics and Space (IAE) of the Department of Aerospace Science and Technology (DCTA). Experience in the development of critical software systems.

Martha Adriana Dias Abdala Master in Applied Computing by the National Institute for Space Research - INPE (2004). Senior Technologist at the Institute of Aeronautics and Space (IAE) of the Department of Aerospace Science and Technology (DCTA). She has experience in Computer Science, with emphasis on Software Engineering.

Selma Shin Shimizu Melnikoff PhD in Electrical Engineering by the Escola Politécnica of the University of São Paulo (1982). Full Professor at the Escola Politécnica of University of São Paulo since 2005. She has experience in Software Engineering, specially in requirements engineering and software architecture.

Mauricio Gonçalves Vieira Ferreira Doctor in Applied Computing by the National Institute for Space Research (2001). Researcher at INPE's Satellite Tracking and Control Center. He is a Full Professor at INPE's Postgraduate Course in Space Engineering and Technology. Member of the International Committee for Standardization of Software in the Space Area (CCSDS). Member of the organizing committee of the international space congress - SPACEOPS. Scientific advisor at FAPESP in the area of Software Engineering.